# 26th International Conference on Algorithms for Bioinformatics

**WABI 2026, August 31–September 2, 2026, L'Aquila, Italy**

Edited by

Nadia El-Mabrouk

Fabio Vandin



*Editors*

**Nadia El-Mabrouk**
DIRO, Université de Montréal, Canada
mabrouk@iro.umontreal.ca

**Fabio Vandin**
University of Padova, Italy
fabio.vandin@unipd.it

*ACM Classification 2012*
Applied computing → Bioinformatics; Applied computing → Computational biology; Applied computing → Computational genomics; Applied computing → Systems biology; Theory of computation → Design and analysis of algorithms; Mathematics of computing; Applied computing → Genetics

**ISBN 978-3-95977-446-8**

*Published online and open access by*
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-446-8.

*Publication date*
August, 2026

*Bibliographic information published by the Deutsche Nationalbibliothek*
The Deutsche Nationalbibliothek lists all publications of this volume in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at https://portal.dnb.de.

*License*
This work is licensed under a Creative Commons Attribution 4.0 International license (CC-BY 4.0): https://creativecommons.org/licenses/by/4.0/legalcode.
In brief, this license authorizes each and everybody to share (to copy, distribute and transmit) the work under the following conditions, without impairing or restricting the authors' moral rights:
- Attribution: The work must be attributed to its authors.

The copyright is retained by the corresponding authors.

Digital Object Identifier: 10.4230/LIPIcs.WABI.2026.0

**ISBN 978-3-95977-446-8** **ISSN 1868-8969** **https://www.dagstuhl.de/lipics**

## LIPIcs – Leibniz International Proceedings in Informatics

LIPIcs is a series of high-quality conference proceedings across all fields in informatics. LIPIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

*Editorial Board*

- Patricia Bouyer-Decitre (Université Paris-Saclay, CNRS, ENS Paris-Saclay, FR)
- Roberto Di Cosmo (Inria and Université Paris Cité, FR)
- Faith Ellen (University of Toronto, CA)
- Holger Hermanns (Universität des Saarlandes, Saarbrücken, DE and Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Wadern, DE)
- Laura Kovács (TU Wien, AT)
- Daniel Král' (Leipzig University, DE and Max Planck Institute for Mathematics in the Sciences, Leipzig, DE – Chair)
- Sławomir Lasota (University of Warsaw, PL)
- Meena Mahajan (Institute of Mathematical Sciences, Chennai, IN)
- Chih-Hao Luke Ong (Nanyang Technological University, SG)
- Eva Rotenberg (IT University of Copenhagen, DK)
- Pierre Senellart (ENS, Université PSL, Paris, France)
- Alexandra Silva (Cornell University, Ithaca, US)

**ISSN 1868-8969**

**https://www.dagstuhl.de/lipics**

# Contents

Preface
*Nadia El-Mabrouk and Fabio Vandin* ..... 0:ix

## Regular Papers

Is Level-1 Blob Reconstruction Under the Network Multispecies Coalescent Easy?
*Junyan Dai and Erin K. Molloy* ..... 1:1–1:24

Improved Approximation Algorithms and Hardness Results for Shortest Common Superstring with Reverse Complements
*Ryosuke Yamano and Tetsuo Shibuya* ..... 2:1–2:13

GSI: A New Approach to the Protein Inference Problem
*Aurélien Berthier, Émile Benoist, Guillaume Fertin, and Géraldine Jean* ..... 3:1–3:16

Contig Model for Variable-Order de Bruijn Graphs
*Diego Díaz-Domínguez, Pierfrancesco Martinello, Taku Onodera, Simon J. Puglisi, and Leena Salmela* ..... 4:1–4:19

10-Minimizers: A Promising Class of Constant-Space Minimizers
*Arseny Shur, Ido Tziony, and Yaron Orenstein* ..... 5:1–5:20

Construction of Distinct $k$-mer Color Sets via Set Fingerprinting
*Jarno N. Alanko and Simon J. Puglisi* ..... 6:1–6:17

Theoretically and Practically Faster Algorithms for Protein Structure Alignment
*Masahito Tsukahara and Tetsuo Shibuya* ..... 7:1–7:13

Efficient Algorithms for Pangenome Personalization
*Denys Andrukhovskyi, Martin Madzin, Luca Denti, Tomáš Vinař, and Broňa Brejová* ..... 8:1–8:17

Quantum Closest-Pair Search for Biological Sequences via k-Mer Distribution Statistics
*Zhezheng Xander Song and Carl Kingsford* ..... 9:1–9:18

Exact and Efficient Inference of Tumor Phylogenies via Novel Pruning Techniques
*Juan Luque, Jacob Gilbert, Arjun Subramanian, Aravind Srinivasan, Salem Malikic, and S. Cenk Sahinalp* ..... 10:1–10:20

FPT Learning of Sparse, Robust and Interpretable Generative Models of RNA Evolution
*Samuel Gardelle, Laurent Bulteau, and Yann Ponty* ..... 11:1–11:22

RNA Inverse Folding Under Stacked Base Pair Maximization
*Théo Boury, Laurent Bulteau, and Yann Ponty* ..... 12:1–12:21

Revisiting $O(n \log \log n)$ Chaining for Anchored Edit Distance
*Nicola Rizzo and Ragnar Groot Koerkamp* ..... 13:1–13:20

Quik 2.0: Efficient Large-Scale DNA Barcode Calling
*Steffen Schüler, Antonia Schmidt, and Matthias Müller-Hannemann* ..... 14:1–14:19

26th International Conference on Algorithms for Bioinformatics (WABI 2026).
Editors: Nadia El-Mabrouk and Fabio Vandin
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Towards a Unified Exact Solution of Rearrangement Small Parsimony for Natural Genomes
*Leonard Bohnenkämper and Daria Frolova* ..... 15:1–15:21

DivQuant: Estimation of Species Richness and Entropy from Small Samples
*Johanna Elena Schmitz and Sven Rahmann* ..... 16:1–16:24

Discriminative Learning of Substitution Matrices and Gap Penalties for Pairwise Alignment of Biological Sequences
*Michał Aleksander Ciach, Elissavet Zacharopoulou, Michał Piotr Startek, Błażej Miasojedow, and Panagiotis Alexiou* ..... 17:1–17:23

Fast Set Operations for Compact $k$-mer Sets
*Jarno N. Alanko, Lore Depuydt, Camille Marchet, and Simon J. Puglisi* ..... 18:1–18:21

On the Complexity of the $(\ell, k)$-Median Problems
*Luís Cunha, Thiago Nascimento, Marilia D. V. Braga, and Jens Stoye* ..... 19:1–19:17

Turnpike with Uncertain Measurements: Triangle-Equality Integer Programming with a Deterministic Recovery Guarantee
*C. S. Elder, Guillaume Marçais, and Carl Kingsford* ..... 20:1–20:20

Designing Exact Spaced Seed Filters Based on Combined Hit and Coverage Information
*Moein Karami, Jens Zentgraf, and Sven Rahmann* ..... 21:1–21:17

The Anti-Lexicographic SUS-Anchor: An Empirically Optimal Selection Scheme
*Ragnar Groot Koerkamp* ..... 22:1–22:15

Statistical Inconsistency of Error-Correction Objectives for Perfect Phylogenies
*Gryte Satas, Matthew A. Myers, and Sohrab P. Shah* ..... 23:1–23:24

CoSTAR: Coarse Stem-Topology Alignment of Pseudoknotted RNA Structures by Relation-Constrained Search
*Finn Archinuk and Hosna Jabbari* ..... 24:1–24:21

Constructing Incompatibility Graphs of Pairs of Trees in Optimal Output-Sensitive Time
*Manuel Lafond* ..... 25:1–25:20

FBApro: A Fast, Simple Linear Transformation for Diverse Metabolic Modeling Tasks
*Ariel Bruner and Mona Singh* ..... 26:1–26:22

Finimap: Fast and Accurate Single-Species Bacterial Pseudoalignment with Finimizers
*Jarno N. Alanko, Elena Biagi, and Simon J. Puglisi* ..... 27:1–27:21

Selecting Chromosomes for Polygenic Traits: Algorithms and Complexity
*Or Zuk* ..... 28:1–28:20

Reconciling and Comparing Variation Graphs Using Homology Relations
*Anna Lisiecka, Adam Cicherski, and Norbert Dojer* ..... 29:1–29:15

PRISM: Partition-Function Decomposition into Structural Classes for Hierarchically Constrained RNA Pseudoknot Ensembles
*Mateo Gray, Sebastian Will, and Hosna Jabbari* ..... 30:1–30:18

## Extended Abstract

Minimum Flow Decomposition Guided by Saturating Subflows
*Ke Chen, Abhishek Talesara, Sanchal Thakkar, and Mingfu Shao* ..... 31:1–31:5

# Preface

This volume comprises the proceedings, extended abstracts, and invited talks presented at the 26th International Conference on Algorithms for Bioinformatics (WABI 2026). The 26th iteration of WABI was held at the Department of Human Studies and at the "San Basilio" Congress Centre of the University of L'Aquila, Italy, from August 31st to September 2nd, 2026.

These proceedings mark the 26th anniversary of the conference, which was established in 2001. The focus of WABI is, and has been, on the introduction of new algorithms, data structures, and methodological approaches and their application to important problems in molecular biology. The emphasis is on algorithms founded on sound models, that are computationally efficient, and that provide evidence of their potential usefulness in practice. The conference provides a venue for presenting methods that improve the state-of-the-art in tackling well-studied problems and for exploring and investigating exciting new ideas and future research directions, including significant works-in-progress. Many interesting and innovative computational approaches, that later see large-scale adoption and application, have previously received their introduction at WABI.

WABI 2026 received 68 manuscript submissions (out of 70 initial abstract submissions), representing a 70% increase compared to last year. Of these submissions, 31 papers were ultimately accepted for inclusion in the conference proceedings. Submissions and reviews were managed using the EasyChair system. Each submission underwent a rigorous review by the program committee, composed of experts, and by the co-reviewers they appointed. Nearly all submissions received three reviews, with an average of 3.13 reviews per submission. A discussion period followed the initial review deadline to reach a decision on each paper, striving to understand the nature of any substantial discrepancies between reviews and to provide authors with the best possible feedback. The range of expertise of the program committee was wide, as were the set of topics of submitted papers and those ultimately accepted for the proceedings, including RNA structure prediction, phylogenetic trees and networks, pangenomics, DNA indexing, sequence and structure alignment, tumor Evolution and more. In addition to the proceedings papers, the conference also welcomed two keynote speakers: Benjamin J. Raphael, from Princeton University, presented "Algorithms for Cell Lineage Tracing: From Trees to Differentiation Maps", and Nadia Pisanti, from the University of Pisa, presented "Algorithmic Toolkits for Pangenomes".

As is customary, selected papers from WABI 2026 have been invited for special consideration, in an extended form, in a thematic series in the journal "Algorithms for Molecular Biology" (AMB), published by BioMed Central.

Finally, we would like to thank all of those without whom this conference would not have been possible. We thank the WABI steering committee for their guidance and feedback in organizing and framing this iteration of the conference, and in helping to tackle challenges that arose. We are also deeply grateful to all of the authors who submitted their exciting work to WABI 2026. This proliferation of research is a source of innovation that advances knowledge in the field of computational biology, creates synergy among members of this research community, and fuels their future projects. Of course, we are also thankful to the outstanding program committee and their nominated sub-reviewers for their substantial efforts in thoroughly reviewing all of the submitted work, and for their diligence in completing their reviews and engaging in fruitful and enlightening discussions that ultimately led to the final selection of the work included in these proceedings.

26th International Conference on Algorithms for Bioinformatics (WABI 2026).
Editors: Nadia El-Mabrouk and Fabio Vandin
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

We thank our keynote speakers, Dr. Benjamin J. Raphael and Nadia Pisanti, for their engaging and enlightening talks. We also extend our gratitude to Davide Bilò, Gianlorenzo D'Angelo, Luciano Gualà, Stefano Leucci, and Guido Proietti, the local conference co-chairs, for all of their efforts in helping to organize, plan, and prepare the venue for the conference. At last, we would like to thank all of the conference participants, speakers and student volunteers, whose ideas, discussions and engagement define the scientific character of the WABI community.

Previous WABI proceedings appeared in LNCS/LNBI volumes as follows: 2149 (WABI 2001, Aarhus), 2452 (WABI 2002, Rome), 2812 (WABI 2003, Budapest), 3240 (WABI 2004, Bergen), 3692 (WABI 2005, Mallorca), 4175 (WABI 2006, Zurich), 4645 (WABI 2007, Philadelphia), 5251 (WABI 2008, Karlsruhe), 5724 (WABI 2009, Philadelphia), 6293 (WABI 2010, Liverpool), 6833 (WABI 2011, Saarbrücken), 7534 (WABI 2012, Ljubljana), (WABI 2013, Sophia Antipolis), 8701 (WABI 2014, Wroclaw), 9289 (WABI 2015, Atlanta), and 9838 (WABI 2016, Aarhus). Beginning in 2017, they appeared in LIPIcs volumes: 88 (WABI 2017, Boston), 113 (WABI 2018, Helsinki), 143 (WABI 2019, Boston), 172 (WABI 2020, held virtually in Pisa), 201 (WABI 2021, held virtually in Chicago), 242 (WABI 2022, Potsdam) and 273 (WABI 2023, Houston), 312 (WABI 2024, Egham, United Kingdom) and 344 (WABI 2025, Maryland, USA).

# WABI 2026 Committees

## Steering Committee

Lenore Cowen
Tufts University, USA

Nadia Pisanti
University of Pisa, Italy

Jens Stoye
Bielefeld University, Germany

Mona Singh
Princeton University, USA

## PC Chairs

Nadia El-Mabrouk
Université de Montréal, Canada

Fabio Vandin
University of Padova, Italy

## Program Committee

Tatsuya Akutsu
Kyoto University, Japan

Jarno Alanko
University of Helsinki, Finland

Lorraine Ayad
Brunel University, London, UK

Broňa Brejová
Comenius University in Bratislava, Slovakia

Yao-Ban Chan
The University of Melbourne, Australia

Panagiotis Charalampopoulos
King's College London, UK

Manuel Cáceres
Aalto University, Finland

Daniel Doerr
Heinrich Heine University Düsseldorf, Germany

Mohammed El-Kebir
University of Illinois at Urbana-Champaign, USA

Jonas Ellert
DIENS, École Normale Supérieure, Paris, France

Anna Gambin
Institute of Informatics, Warsaw University, Poland

Ragnar Groot Koerkamp
ETH Zurich, Switzerland

Momoko Hayamizu
Waseda University, Japan

Hosna Jabbari
University of Alberta, Canada

Gregory Kucherov
CNRS/LIGM, France

Manuel Lafond
Université de Sherbrooke, Canada

Benjamin Langmead
Johns Hopkins University, USA

Yuri Pirola
DISCo, Univ. degli Studi di Milano-Bicocca, Italy

Solon Pissis
The Cyprus Institute, Cyprus

Cinzia Pizzi
University of Padova, Italy

Nicola Prezza
Ca' Foscari University, Italy

Sven Rahmann
Saarland University, Germany

Leena Salmela
University of Helsinki, Finland

26th International Conference on Algorithms for Bioinformatics (WABI 2026).
Editors: Nadia El-Mabrouk and Fabio Vandin
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Mingfu Shao
Carnegie Mellon University, USA

Wing-Kin Sung
Chinese University of Hong Kong, China

Nadia Tahiri
Université de Sherbrooke, Canada

Sharma Thankachan
North Carolina State University, Raleigh, USA

Simone Zaccaria
UCL Cancer Institute, UK

Meirav Zehavi
Ben-Gurion University, Israel

Jie Zheng
ShanghaiTech University, China

## Subreviewers

Ajmain Yasar Ahmed
Jarno Alanko
Eloi Araujo
Rocco Ascone
Anton Bankevich
Carl Barton
Albert Bifet
Maria Bochenek
Itai Boneh
Magnus Bordewich
Edward Braun
Paolo Bresolin
Karel Brinda
Nathaniel Brown
Sebastian Bruchhold
Alessio Campanelli
Davide Cenzato
Carmen Cerchia
Cedric Chauve
Ke Chen
Adam Cicherski
Alessio Conte
Fawaz Dabbaghie
Sarath Dantu
Mattéo Delabre
Alex Della Schiava
Diego Díaz-Domínguez
Maksims Dimitrijevs
Barbara Domżał
C. S. Elder
Victor Epain
Parsa Eskandar
Massimo Equi
Carlo Ferrari
Guillaume Fertin
Gabriele Fici
Martin Frith
Estéban Gabory
Askar Gafurov
Arnab Ganguly
Mathieu Gascon
Daniel Gibney
Antoni Goldstein
Pawel Gorecki
Sayan Goswami
Veronica Guerrini
Valeriia Haberland
Sean Hackett
Juha Harviainen
Peter Heringer
Katharina Huber
Stephen Hwang
Chirag Jain
Daniel Jones
Irtesam Mahmud Khan
Benny Kimelfeld
Tomasz Kociumaka
Zofia Kozaryna
Aravind Krishnan A
Justyna Król
Émeric Laberge
Kaari Landry
Simone Linz
Bo Liu
Michael Love
Grigorios Loukides
Magdalena Machnicka
Rahul Nihalani
Aaryan Mahesh Sarnaik
Pierfrancesco Martinello
Francesco Masillo
Paul Medvedev
Joao Meidanis
Pall Melsted
Irmtraud M. Meyer
Marcin Mucha
Rahul Nihalani
Laurent Noé
Bernhard Palsson
Katarzyna Paluch
Luca Parmigiani
Jarosław Paszek
Benedict Paten
Loïc Paulevé
Leonardo Pellegrina
Rebecca Pfeil
Giulio Ermanno Pibiri
Barbara Poszewiecka
Mano Prakash Parthasarathi
Simon Puglisi
Daniel Puttini
Nicola Rizzo
Mark Robinson
Giovanna Rosone
Eytan Ruppin
Leena Salmela
Johanna Elena Schmitz
Daniel Segre
Francisco Sena
Tomer Shlomi
Jouni Sirén
Megan Smith
Giltae Song
Dominik Stanojevic
Katherine St John

Michelle Sweering
Yuki Takazawa
Carlo Tosoni
Kavya Vaddadi
Giorgio Venturin
Pavel Veselý
Tomas Vinar
Tomasz Walen
Lizhi Wang
Yadong Wang
Tandy Warnow
Martin Weigt
Taoyang Wu
Yi Xing
Jinbo Xu
Tasfia Zahin
Alexander Zelikovsky
Jens Zentgraf
Fangqing Zhao
Binhai Zhu

# List of Authors


Jarno N. Alanko (6, 18, 27)
Department of Computer Science, University of Helsinki, Finland

Panagiotis Alexiou (17)
Department of Applied Biomedical Science, Faculty of Health Sciences, University of Malta, Msida, Malta

Denys Andrukhovskyi (8)
Faculty of Mathematics, Physics and Informatics, Comenius University Bratislava, Slovakia

Finn Archinuk (24)
Department of Biomedical Engineering, University of Alberta, Edmonton, Canada

Émile Benoist (3)
Nantes Université, École Centrale Nantes, CNRS, LS2N, UMR 6004, F-44000 Nantes, France

Aurélien Berthier (3)
Nantes Université, École Centrale Nantes, CNRS, LS2N, UMR 6004, F-44000 Nantes, France

Elena Biagi (27)
Department of Computer Science, University of Helsinki, Finland

Leonard Bohnenkämper (15)
Faculty of Technology and Center for Biotechnology (CeBiTec), Bielefeld University, Germany

Théo Boury (12)
LIX, CNRS, École polytechnique, Institut Polytechnique de Paris, Palaiseau, France

Marilia D. V. Braga (19)
Faculty of Technology and Center for Biotechnology (CeBiTec), Bielefeld University, Germany

Broňa Brejová (8)
Faculty of Mathematics, Physics and Informatics, Comenius University Bratislava, Slovakia

Ariel Bruner (26)
Princeton University, NJ, USA

Laurent Bulteau (11, 12)
LIGM, CNRS, Université Gustave Eiffel, Marne-la-vallée, France

Ke Chen (31)
Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA, USA

Michał Aleksander Ciach (17)
Department of Applied Biomedical Science, Faculty of Health Sciences, University of Malta, Msida, Malta

Adam Cicherski (29)
Institute of Informatics, University of Warsaw, Poland

Luís Cunha (19)
Instituto de Computação, Universidade Federal Fluminense, Niterói, Brazil

Junyan Dai (1)
Department of Computer Science, University of Maryland, College Park, MD, USA

Luca Denti (8)
Faculty of Mathematics, Physics and Informatics, Comenius University Bratislava, Slovakia

Lore Depuydt (18)
Univ. Lille, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France

Norbert Dojer (29)
Institute of Informatics, University of Warsaw, Poland

Diego Díaz-Domínguez (4)
Department of Computer Science, University of Helsinki, Finland

C. S. Elder (20)
Ray and Stephanie Lane Computational Biology Department, Carnegie Mellon University, Pittsburgh, PA, USA

Guillaume Fertin (3)
Nantes Université, École Centrale Nantes, CNRS, LS2N, UMR 6004, F-44000 Nantes, France

Daria Frolova (15)
European Bioinformatics Institute, Hinxton, Cambridge, UK; Department of Computational Biology, University of Lausanne, Switzerland

Samuel Gardelle (11)
LIX, CNRS, École polytechnique, Institut Polytechnique de Paris, Palaiseau, France



26th International Conference on Algorithms for Bioinformatics (WABI 2026).
Editors: Nadia El-Mabrouk and Fabio Vandin
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Jacob Gilbert (10)
Cancer Data Science Laboratory, National Cancer Institute, Bethesda, MD, USA

Mateo Gray (30)
Department of Biomedical Engineering, University of Alberta, Edmonton, Canada

Ragnar Groot Koerkamp (13, 22)
Karlsruhe Institute of Technology, Germany

Hosna Jabbari (24, 30)
Department of Biomedical Engineering, University of Alberta, Edmonton, Canada

Géraldine Jean (3)
Nantes Université, École Centrale Nantes, CNRS, LS2N, UMR 6004, F-44000 Nantes, France

Moein Karami (21)
Algorithmic Bioinformatics, Faculty of Mathematics and Computer Science, Saarland University, Saarbrücken, Germany; Center for Bioinformatics Saar, Saarland Informatics Campus, Saarbrücken, Germany; Saarbrücken Graduate School of Computer Science, SIC, Saarbrücken, Germany

Carl Kingsford (9, 20)
Ray and Stephanie Lane Computational Biology Department, Carnegie Mellon University, Pittsburgh, PA, USA

Manuel Lafond (25)
Université de Sherbrooke, Canada

Anna Lisiecka (29)
Institute of Informatics, University of Warsaw, Poland

Juan Luque (10)
Department of Computer Science, University of Maryland, College Park, MD, USA

Martin Madzin (8)
Faculty of Mathematics, Physics and Informatics, Comenius University Bratislava, Slovakia

Salem Malikic (10)
Cancer Data Science Laboratory, National Cancer Institute, Bethesda, MD, USA

Camille Marchet (18)
Univ. Lille, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France

Pierfrancesco Martinello (4)
Department of Computer Science, University of Helsinki, Finland

Guillaume Marçais (20)
Ray and Stephanie Lane Computational Biology Department, Carnegie Mellon University, Pittsburgh, PA, USA

Błażej Miasojedow (17)
Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland

Erin K. Molloy (1)
Department of Computer Science, University of Maryland, College Park, MD, USA; University of Maryland Institute for Advanced Computer Studies, College Park, MD, USA

Matthew A. Myers (23)
Halvorsen Center for Computational Oncology, Memorial Sloan Kettering Cancer Center, New York, NY, USA

Matthias Müller-Hannemann (14)
Martin Luther University Halle-Wittenberg, Germany

Thiago Nascimento (19)
Instituto de Computação, Universidade Federal Fluminense, Niterói, Brazil

Taku Onodera (4)
Graduate School of Information Science and Technology, University of Tokyo, Japan

Yaron Orenstein (5)
Department of Computer Science and The Mina and Everard Goodman Faculty of Life Sciences, Bar-Ilan University, Ramat Gan, Israel

Yann Ponty (11, 12)
LIX, CNRS, École polytechnique, Institut Polytechnique de Paris, Palaiseau, France

Simon J. Puglisi (4, 6, 18, 27)
Department of Computer Science, University of Helsinki, Finland

Sven Rahmann (16, 21)
Algorithmic Bioinformatics, Faculty of Mathematics and Computer Science, Saarland University, Saarbrücken, Germany; Center for Bioinformatics Saar, Saarland Informatics Campus, Saarbrücken, Germany

Nicola Rizzo (13)
University of Helsinki, Finland

S. Cenk Sahinalp (10)
Cancer Data Science Laboratory, National Cancer Institute, Bethesda, MD 20894, USA

Leena Salmela (4)
Department of Computer Science, University of Helsinki, Finland

Gryte Satas (23)
Halvorsen Center for Computational Oncology, Memorial Sloan Kettering Cancer Center, New York, NY, USA

Antonia Schmidt (14)
Martin Luther University Halle-Wittenberg, Germany

Johanna Elena Schmitz (16)
Algorithmic Bioinformatics, Faculty of Mathematics and Computer Science, Saarland University, Saarbrücken, Germany; Saarbrücken Graduate School of Computer Science, Germany; Center for Bioinformatics Saar, Saarland Informatics Campus, Saarbrücken, Germany

Steffen Schüler (14)
Martin Luther University Halle-Wittenberg, Germany

Sohrab P. Shah (23)
Halvorsen Center for Computational Oncology, Memorial Sloan Kettering Cancer Center, New York, NY, USA

Mingfu Shao (31)
Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA, USA; Huck Institutes of the Life Sciences, The Pennsylvania State University, University Park, PA, USA

Tetsuo Shibuya (2, 7)
Division of Medical Data Informatics, Human Genome Center, Institute of Medical Science, The University of Tokyo, Japan

Arseny Shur (5)
Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel

Mona Singh (26)
Princeton University, NJ, USA

Zhezheng Xander Song (9)
Ray and Stephanie Lane Computational Biology Department, Carnegie Mellon University, Pittsburgh, PA, USA

Aravind Srinivasan (10)
Department of Computer Science, University of Maryland, College Park, MD, USA

Michał Piotr Startek (17)
Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland

Jens Stoye (19)
Faculty of Technology and Center for Biotechnology (CeBiTec), Bielefeld University, Germany

Arjun Subramanian (10)
The Pingry School, Basking Ridge, NJ, USA

Abhishek Talesara (31)
Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA, USA

Sanchal Thakkar (31)
Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA, USA

Masahito Tsukahara (7)
Human Genome Center, The Institute of Medical Science, The University of Tokyo, Japan; Department of Computer Science, Graduate School of Information Science and Technology, The University of Tokyo, Japan

Ido Tziony (5)
Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel

Tomáš Vinař (8)
Faculty of Mathematics, Physics and Informatics, Comenius University Bratislava, Slovakia

Sebastian Will (30)
Department of Computer Science, Institut Polytechnique de Paris, Palaiseau, France

Ryosuke Yamano (2)
Department of Computer Science, Graduate School of Information Science and Technology, The University of Tokyo, Japan; Division of Medical Data Informatics, Human Genome Center, Institute of Medical Science, The University of Tokyo, Japan

Elissavet Zacharopoulou (17)
Department of Applied Biomedical Science, Faculty of Health Sciences, University of Malta, Msida, Malta

Jens Zentgraf (21)
Algorithmic Bioinformatics, Faculty of Mathematics and Computer Science, Saarland University, Saarbrücken, Germany; Center for Bioinformatics Saar, Saarland Informatics Campus, Saarbrücken, Germany; Saarbrücken Graduate School of Computer Science, SIC, Saarbrücken, Germany

Or Zuk (28)
Department of Statistics and Data Science, The Hebrew University of Jerusalem, Israel

# Is Level-1 Blob Reconstruction Under the Network Multispecies Coalescent Easy?

**Junyan Dai**
Department of Computer Science, University of Maryland, College Park, MD, USA

**Erin K. Molloy**[1]
Department of Computer Science, University of Maryland, College Park, MD, USA
University of Maryland Institute for Advanced Computer Studies, College Park, MD, USA

## Abstract

Hybridization is an important evolutionary process, commonly modeled by the network multispecies coalescent. Reconstructing evolutionary histories under this model is notoriously costly, even for level-1 networks where hybridization events are isolated from each other. The widely used methods that combine speed with statistical guarantees rely on quartet concordance factors computed for all subsets of four species, resulting in an $O(n^4k)$ bottleneck that severely limits scalability to large numbers of species ($n$) and genes ($k$). Among quartet-based methods, NANUQ+ is notable because it decomposes the problem into two steps: first reconstructing a *tree of blobs*, which compresses each non-treelike part of the network, called a blob, into a single vertex, and second reconstructing the internal structure of each level-1 blob, specifically its circular order and hybrid vertex. Here, we investigate whether level-1 blob reconstruction is difficult once the tree of blobs is known. We present a fast and statistically consistent algorithm, called NetCS, based on two simple primitives: majority voting and merge sort, circumventing the bottleneck of computing all quartet concordance factors. In simulations, NetCS achieved comparable accuracy to NANUQ+ and was dramatically faster, enabling analyses of 200 taxa and 1000 genes in only a few minutes. Both methods attained near-perfect accuracy when given the true tree of blobs; however, their performance degraded in end-to-end pipelines due to errors in tree of blobs reconstruction. Strikingly, even methods that reconstruct level-1 networks directly struggled to accurately predict hybrid ancestry. Our results suggest that reconstructing level-1 blobs is unexpectedly easy once the tree of blobs is known, and that a major challenge for phylogenetic network inference lies in accurate tree of blobs reconstruction.

**2012 ACM Subject Classification** Applied computing → Molecular evolution

**Keywords and phrases** Phylogenetic Networks, Circular Orders, Quartets, Network Multispecies Coalescent

**Digital Object Identifier** 10.4230/LIPIcs.WABI.2026.1

**Supplementary Material** *Software (NetCS)*: `https://github.com/molloy-lab/TREE-QMC` archived at `swh:1:dir:b2c6ac29f14205f67d37a473941cfd7d288270d1`

**Funding** This material is based upon work supported by the U. S. National Science Foundation (NSF) under Grant No. 2441458 (to EKM) as well as the State of Maryland. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the sponsors.

## 1 Introduction

Hybridization and admixture are important evolutionary scenarios that must be represented as phylogenetic networks rather than trees. A phylogenetic network governs the ancestry of unlinked, recombination-free genomic regions, called genes, commonly modeled by the

[1] Corresponding author

© Junyan Dai and Erin K. Molloy;
licensed under Creative Commons License CC-BY 4.0

Network Multispecies Coalescent (NMSC) [40, 41]. Reticulations in the network enable genes from the same species' genome to trace back to different ancestral populations, resulting in conflicting genealogies, called gene trees. Additionally, conflicts among gene trees can arise from incomplete lineage sorting (ILS), similar to the traditional MSC model [29]. Although gene trees are not directly observed, they govern the evolution of DNA sequences, from which they can be estimated with fast methods, such as RAxML [36] and IQ-TREE [26].

Reconstructing phylogenetic networks under the NMSC is notoriously costly even for level-1 networks where hybridization events are isolated from each other. The popular methods that combine speed with statistical guarantees rely on *quartet concordance factors (qCFs)* computed for all subsets of four species, called taxa. A quartet is an unrooted phylogenetic tree on four species, and its concordance factor is simply the number of gene trees that have the same topology after restriction to the same species. Performing this precomputation for all four-taxon subsets results in an $O(n^4k)$ cost that severely limits scalability to large numbers of species ($n$) and genes ($k$), although methods based on qCFs are still significantly cheaper than those that based on full likelihood calculations (e.g., [41]).

The first network reconstruction method to leverage qCFs was SNaQ [34, 35], which conducts a search over the space of (semi-directed) level-1 networks via edit moves, seeking a network to maximize its pseudo-likelihood function (see [24] for a related composite likelihood function for DNA sequences). To improve scalability beyond $\sim$30 species, the most recent version of SNaQ attempts to approximate pseudo-likelihood by randomly sampling four-taxon subsets [22]; this approach currently has no statistical guarantees, and there are intricacies to random sampling, as highlighted by [10]. Alternatively, SNaQ can be leveraged within the divide-and-conquer framework InPhyNet [23], but then the goal is to run SNaQ on pairwise disjoint subsets of taxa that are as large as possible.

Another recently developed method that leverages qCFs is CAMUS [38], accepted to ISMB 2026. Unlike SNaQ, CAMUS takes a rooted base tree as input (typically estimated via ASTRAL-IV [43]) and then adds reticulations edges to maximize the number of quartets also displayed by the (directed) level-1 network (also see [28]). Although CAMUS is more scalable than SNaQ, it still requires qCF precomputation. CAMUS does not currently have statistical guarantees under the NMSC and can be statistically inconsistent when ASTRAL-IV is used to construct the base tree, as recently shown by Dinh and Baños [11] (similar issues for tree-based network inference arise in the context of admixture graph reconstruction [27]).

In contrast, NANUQ+ [4] decomposes (semi-directed) level-1 network reconstruction into two steps: (1) reconstructing a tree of blobs, which compresses each non-treelike part of the network, called a blob, into a single vertex and then (2) reconstructing the internal structure of each level-1 blob, specifically its circular order and hybrid vertex (also see [16]). Each step as implemented by NANUQ+ is provably *statistically consistent* under the NMSC with some assumptions [3, 4]; however, both steps require qCF precomputation, limiting scalability.

Recent work has focused on scalable tree of blobs reconstruction from qCFs with statistical and/or correctness guarantees (e.g., [3, 10, 17, 13, 30]). This motivates us to investigate whether level-1 blob reconstruction is difficult once the tree of blobs is known. The remainder of the paper is organized as follows. We present the terminology and background in Section 2. In Section 3, we present our new algorithm for level-1 blob reconstruction from qCFs, called NetCS, along with its theoretical properties. NetCS is based on two simple primitives: majority voting and sorting; thus it differs dramatically from NANUQ+'s approach, which computes distances between all pairs of taxa based on the input qCFs and then evaluates the goodness of fit between expected and observed distances for all possible level-1 blob structures.

In Section 4, we describe the results of benchmarking NetCS against NANUQ+ and CAMUS. We conclude in Section 5 with a discussion of whether level-1 blob reconstruction is difficult based on our theoretical and empirical results.

## 2 Background

To present Net-CS, we review terminology and background for phylogenetic networks.

**Phylogenetic Networks.** A *directed* phylogenetic network $N^+$ is a directed simple acyclic finite graph with the property that there exists a directed path from the root, a unique vertex with in-degree 0, to all other vertices. The leaf vertices of $N^+$, which have out-degree 0, are bijectively labeled by a set $S$ of species. A *hybrid* is a vertex with in-degree $> 1$, and a *reticulation edge* is an arc whose child is a hybrid. A *wedge* path in $N^+$ is a sequence of distinct vertices $u_0, u_1, \ldots, u_{k-1}, u_k$ such that there exists some vertex $u_i$ that "roots" the path so $u_i, u_{i-1}, \ldots, u_2, u_1$ and $u_i, u_{i+1}, \ldots, u_{k-1}, u_k$ are both directed paths in $N^+$ [17]. A directed network $N^+$ can be converted into a *semi-directed* network $N^-$ [34] by removing all edges and vertices above the lowest stable ancestor $s$ of $S$ (i.e., the lowest vertex on all directed paths from the root to each leaf in $S$; see [37], page 263), undirecting all non-reticulation edges, and suppressing $s$ if it has degree 2. The wedge path terminology is used for semi-directed networks because there is a bijection between wedge paths in $N^+$ and paths in $N^-$ that do not traverse two reticulation edges incident to the same hybrid vertex. If there are no hybrid vertices, $N^-$ is simply an unrooted phylogenetic tree. We assume that semi-directed networks and unrooted trees are *binary*, specifically leaves have degree 1 and all other vertices have degree 3.

**Cut Edges, Blobs, and Partitions.** A bridge, also called a *cut edge*, is an edge whose deletion increases the number of connected components of the graph. The maximal bridgeless subgraphs of $N^-$, when ignoring the directionality of reticulation edges, are called *blobs*. The *level* of a blob is the number of reticulation edges in it minus the number of hybrid vertices in it; we assume that all blobs are level-1. The degree of a blob is the number of cut edges adjacent to it. An $m$-blob is a blob with degree $m$. The *tree of blobs (TOB)* [15] is obtained from $N^-$ by contracting the edges in each blob into a single vertex and then suppressing any degree-2 vertices (Fig. 1A–B). Let $v_{\mathcal{B}}$ denote the vertex in the tree of blobs $T_N$ that corresponds to blob $\mathcal{B}$ in the underlying network $N^-$. The deletion of vertex $v_{\mathcal{B}}$ from $T_N$ produces a collection of rooted subgraphs whose leaf sets partition the taxon set $S$, denoted $Part(v_{\mathcal{B}}) = P_1|P_2|\ldots|P_{m_{\mathcal{B}}}$, where $m_{\mathcal{B}}$ is the degree of $v_{\mathcal{B}}$; this is equivalent to the partition induced by deleting all edges and vertices in blob $\mathcal{B}$ from $N^-$, denoted $Part(\mathcal{B})$. Likewise, the deletion of a cut edge $e$ from $N^-$ or its TOB yields two rooted subgraphs whose leaves partition $S$ into two subsets, called a *bipartition* or *split*, denoted $A|B$. A bipartition is trivial if either $|A| = 1$ or $|B| = 1$. If both endpoints of $e$ have degree 3, then deleting $e$, along with its endpoints, induces a *quadrapartition*: $Quad(e) = A_1|A_2|B_1|B_2$.

**Blob structure.** Each blob $\mathcal{B}$ in a binary, semi-directed, level-1 network is either a single vertex or a cycle when ignoring the directionality of reticulation edges; the former is called trivial and the latter non-trivial. Moreover, no two blobs can share an edge or vertex, and each non-trivial blob contains exactly one hybrid vertex incident to exactly two reticulation edges. The **circular order** of $\mathcal{B}$ is the cyclic ordering of vertices encountered when traversing the cycle ignoring edge direction, considered up to rotation and reversal. There is a bijection

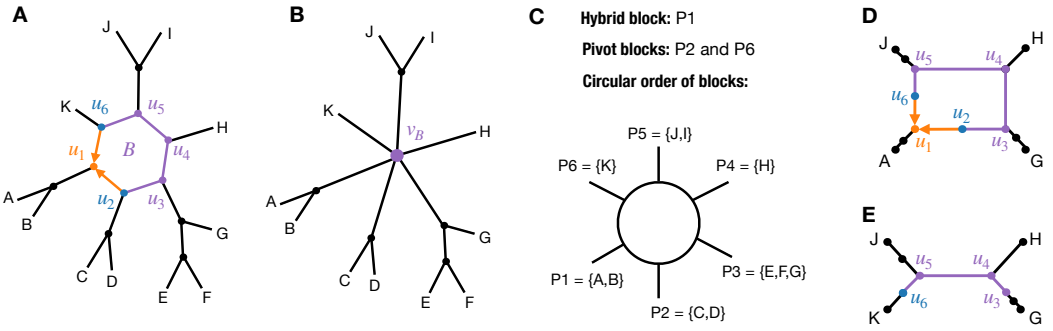



**Figure 1** **A)** A binary, semi-directed, level-1 network $N^-$ on species set $S$ with one non-trivial blob $\mathcal{B}$. **B)** Tree of blobs for $N^-$. Blob vertex $v_{\mathcal{B}}$ induces an 6-partition of $S$ denoted $Part(v_{\mathcal{B}}) = P_1|\ldots|P_6$. **C)** Information about $\mathcal{B}$. **D)** Subnetwork on 4 taxa from distinct blocks of $Part(v_{\mathcal{B}})$, including hybrid block $P_1$. The subnetwork displays two quartets: $A, J|H, G$ and $A, G|H, J$. **E)** Subnetwork on 4 taxa from distinct blocks of $Part(v_{\mathcal{B}})$, excluding hybrid block $P_1$.

between the vertices in the circular order of blob $\mathcal{B}$ and the blocks of $Part(\mathcal{B})$; thus, we refer to them interchangeably. $N^-$ can be reconstructed from its TOB by (1) determining the circular order of each non-trivial blob, and (2) identifying the hybrid of each non-trivial blob, as this gives the directionality of reticulation edges (Fig. 1C) [4].

**Quarnets and quartets.** A *subnetwork* of $N^-$ on $Y \subseteq S$, denoted $N^-|_Y$, is the subgraph induced by the union of vertices and edges on all wedge paths in $N^-$ between all pairs of taxa in $Y$ [5]. The *restriction* of $N^-$ to $Y \subseteq S$ is formed by computing the subnetwork on $Y$ and then repeatedly suppressing degree-2 vertices, parallel arcs, and 2-blobs [17]. For $|Y| = 4$, subnetworks are referred to as **quarnets**. **Quarnet-splits** are the set of bipartitions induced by the cut edges of $N^-|_Y$ or equivalently the edges in the TOB of $N^-|_Y$ [17]. A quarnet either contains a 4-blob or not [3, 17]. In the former case, the TOB of $N^-|_Y$ is an (unresolved) star tree (Fig. 1D), and in the latter case, the TOB of $N^-|_Y$ is a (binary) **quartet** tree (Fig. 1E); thus, we say that the evolutionary relationship is "treelike" [3]. A quartet on species set $\{a, b, c, d\}$ has one of three possible topologies, indicated by the one non-trivial bipartition: $a, b|c, d$ or $a, c|b, d$, or $a, d|b, c$. Any quarnet that contains a 4-blob **displays** exactly two quartets formed by deleting each of the reticulation edges incident to the hybrid vertex given our assumption that $N^-$ is binary and level-1 [34, 7].

**Model and Data.** The Network Multispecies Coalescent (NMSC) model is parameterized by $\mathcal{N}^+ = (N^+, S, \Theta)$, where $N^+$ is a directed phylogenetic network on species set $S$ and $\Theta$ is a set of numerical parameters: branch lengths in coalescent units, reticulation edge inheritance proportions, and the inheritance correlation $\rho$ (note that we assume independent inheritance, i.e., $\rho = 0$). We only briefly summarize the NMSC here and refer readers to [34,10] for a complete description. A gene genealogy is generated under the NMSC by sampling one gene per species and tracing ancestry backward in time until all lineages coalesce (i.e., trace back to a common ancestor), forming a rooted, binary tree on $S$ [40, 12]. We let $\mathbb{P}_{\mathcal{N}^+}(t)$ denote the probability of an unrooted, binary gene tree $t$ on species set $Y \subseteq S$ given $\mathcal{N}^+$. For $|Y| = 4$, $\mathbb{P}_{\mathcal{N}^+}(q_i^Y)$, where $i = \{1, 2, 3\}$ indexes the three possible quartets on $Y$, are the *expected* **quartet concordance factors (qCFs)**. Expected qCFs are fully determined by the semi-directed version of $\mathcal{N}^+$, denoted $\mathcal{N}^-$ (Proposition 3 in [34]). *Observed* qCFs are

computed by counting the number of $k$ *gene trees* whose restriction to $Y$ is isomorphic to $q_i^Y$, denoted $w_k(q_i^Y)$, and then dividing each element in the 3-vector by its sum. Observed qCFs converge simultaneously to their expected values with probability 1 as the number of gene trees goes to infinity by the strong law of large numbers.

**Assumptions and Identifiability.** We make standard assumptions about the model network; see [34, 7, 3, 4, 10]. Specifically, we assume that $\mathcal{N}^-$ is binary, semi-directed, level-1, and metric (i.e., non-reticulation edges have positive lengths, reticulation edges have lengths greater than or equal to zero, reticulation edges incident to the same target have inheritance proportions that sum to 1 and take on values between 0 and 1, non-inclusive). Under these assumptions, the structure of 3-blobs is not identifiable from qCFs [7]. However, the circular order of 4-blobs is identifiable, and for large blobs with degree $> 4$, the hybrid vertex also is identifiable (Theorem 4 in [7]). The crux behind this identifiability result is that for any 4-taxon subset $Y$ such that $N^-|_Y$ contains a 4-blob, the qCFs on $Y$ are distinct, i.e., $\mathbb{P}_{\mathcal{N}^-}(q_1^Y) \neq \mathbb{P}_{\mathcal{N}^-}(q_2^Y) \neq \mathbb{P}_{\mathcal{N}^-}(q_3^Y)$ (Def. 17 and Theorem 3 in [7]). Moreover, the lowest CF corresponds to the only quartet that is not displayed by $N^-|_Y$ (Proposition 11 in [7]). These two properties hold under generic numerical parameter settings except those in a set of measure zero (an example in the set of measure zero can be obtained by setting all edges in Fig. 1A to have the same length in coalescent units and both inheritance proportions to 0.5).

**Oracle for 4-blobs.** Lastly, our approach requires an oracle that takes a 4-taxon subset on $Y$ as input, specifically its associated 3-vector of qCFs, and returns whether the quarnet $N^-|_Y$ contains a 4-blob or not. In practice, we employ the same hypothesis testing framework used for tree of blobs reconstruction in NANUQ+ [4, 3, 2]. There are two versions of the hypothesis test: T3 and cut. The null hypothesis of both is that the $N^-|_Y$ does not contain a 4-blob, thus low $p$-values indicate the null should be rejected (i.e., $N^-|_Y$ does contain a 4-blob). The test assumes that expected qCFs are distinct for every 4-taxon subset $Y \subseteq S$ such that $N^-|_Y$ contains a 4-blob [3], which is satisfied as noted in the previous section. However, the T3 version of the hypothesis test additionally assumes that networks are *class-1 quartet-nonanomalous*, that is, for every 4-taxon subset $Y \subseteq S$ such that $N^-|_Y$ does not contain a 4-blob, $\mathbb{P}_{\mathcal{N}^-}(q_1^Y) > \mathbb{P}_{\mathcal{N}^-}(q_2^Y)$ and $\mathbb{P}_{\mathcal{N}^-}(q_1^Y) > \mathbb{P}_{\mathcal{N}^-}(q_3^Y)$, where $q_1^Y$ corresponds to the TOB of $N^-|_Y$ [3]. This assumption is satisfied for common inheritance ($\rho = 1$) but not necessarily independent inheritance ($\rho = 0$) [5], although strong empirical performance of the T3 option is still possible even under model violations [3, 10].

## 3 Methods

We are now ready to present our method NetCS, which has the following input and output.

- **Input:**
  - Unrooted tree of blobs $T_N$ on a set $S$ of $n$ species
  - Set $\{G_1, \ldots, G_k\}$ of unrooted, binary gene trees, each on species set $S_i$, where $S = \cup_{i=1}^k S_i$
- **Output:** A binary, semi-directed, level-1 network on $S$ with TOB isomorphic to $T_N$ (note that only non-trivial blobs of degree at least 5 are reconstructed)

NetCS processes each vertex with degree 5 or more in $T_N$ *independently* with two primitives: majority voting to detect the hybrid and sorting to resolve the circular order.

## 3.1 Hybrid Majority Voting

Consider some vertex $v_{\mathcal{B}}$ in the input TOB $T_N$ with degree $m_{\mathcal{B}} \geq 5$. The first step in NetCS is to find the hybrid block of $Part(v_{\mathcal{B}})$, which is the unique block induced by the cut edge incident to the reticulation vertex. Our approach is based on the following result.

▶ **Lemma 1.** *Let $\mathcal{N}^- = (N^-, S, \Theta)$ be a binary, semi-directed, level-1, metric species network. Let $\mathcal{O}_k(Y)$ denote the p-value of NANUQ+ hypothesis test applied to some 4-taxon subset $Y \subseteq S$, specifically the qCFs computed from $k$ independent (unrooted) gene trees on species set $S$ sampled from the distribution induced by the NMSC given $\mathcal{N}^-$. Let $v_{\mathcal{B}}$ be a vertex in the TOB for $N^-$ with degree $m_{\mathcal{B}} \geq 5$. Now consider two 4-taxon subsets constructed such that every taxon is drawn from distinct blocks of $Part(v_{\mathcal{B}})$, with the first subset $Y_{+h}$ drawing one taxon from the hybrid block of $Part(v_{\mathcal{B}})$ and the second subset $Y_{-h}$ drawing zero taxa from the hybrid block. Then, with probability $\to 1$ as $k \to \infty$, $\mathcal{O}_k(Y_{+h}) < \mathcal{O}_k(Y_{-h})$ for all pairs of 4-taxon subsets $Y_{+h}, Y_{-h}$ constructed as described above.*

**Proof.** For every pair of 4-taxon subsets $Y_{+h}, Y_{-h}$ constructed as described above, $N|_{Y_{+h}}$ contains a 4-blob but $N|_{Y_{-h}}$ does not, by Lemma 5 in [10]. The remainder of the proof follows from the statistical consistency of the hypothesis test $\mathcal{O}_k$; specifically, Theorem 4 in [3] states that there exists a sequence of significance thresholds $\alpha_k \to 0$ such that the hypothesis test applied to $k$ gene trees will, with probability tending to 1 as $k \to \infty$, return the correct result for all 4-taxon subsets of $S$ (also see [30]). This implies the returned $p$-value will be less than $\alpha_k$ for any 4-taxon subset $Y$ such that $N|_Y$ contains a 4-blob and will be greater than or equal to $\alpha_k$ otherwise, giving us our result. ◀

The innovation behind NetCS is applying the hypothesis test $\mathcal{O}_k$ to carefully selected 4-taxon subsets so that the precomputation of qCFs for all 4-taxon subsets of $S$ can be avoided. Specifically, we create *re-quadrapartitions* of $Part(v_{\mathcal{B}})$ and then apply $\mathcal{O}_k$ to four taxa drawn from distinct blocks.

▶ **Definition 1** (Re-quadraparition). *Let $\mathcal{P} = P_1|P_2|\dots|P_m$ be a $m$-partition with $m \geq 5$. A* re-quadrapartition *is a coarsening of $\mathcal{P}$ into exactly 4 blocks, denoted $P'_1|P'_2|P'_3|P'_4$, such that each original block $P_i$ is contained entirely within exactly one new block $P'_j$.*

Given a re-quadrapatition of $Part(v_{\mathcal{B}})$, denoted $P'_1|P'_2|P'_3|P'_4$, the hypothesis test $\mathcal{O}_k$ is applied within the **3-fix, 1-alter (3f1a) algorithm**, introduced at RECOMB 2026 [10]:

1. Select one taxon arbitrarily from each block of the requadrapartition: $x_1 \in P'_1, x_2 \in P'_2, x_3 \in P'_3, x_4 \in P'_4$.
2. Apply $\mathcal{O}_k$ to the qCFs for $\{x_a, x_2, x_3, x_4\}$ for all $x_a \in P'_1$.
3. Repeat step (2) three times but instead of fixing the three taxa from $P'_1, P'_2, P'_3$ and rotating through all taxa in $P'_1$, swap out $P'_1$ with $P'_2, P'_3, P'_4$.
4. Return the minimum $p$-value found from all applications of $\mathcal{O}_k$, along with the associated minimizer 4-taxon subset.

Unlike its original application, we leverage 3f1a for voting, allowing each taxon in the minimizer subset vote for its respective block of $Part(v_{\mathcal{B}})$. This process is repeated for $c$ rounds, each corresponding to an arbitrary re-quadrapartition of $Part(v_{\mathcal{B}})$. The block of $Part(v_{\mathcal{B}})$ with the most votes is selected as the hybrid, with some additional steps for tie breaking (Algorithm 1); the tie breaking step is closely related to hybrid identifiability [7, 13].

**Algorithm 1** Hybrid Majority Voting Algorithm.

**Input :** Partition of species set $S$ induced by vertex $v_{\mathcal{B}}$, denoted $Part(v_{\mathcal{B}}) = P_1|P_2|\ldots|P_{m_{\mathcal{B}}}$, Set of $k$ unrooted, binary gene trees each on (a subset of) $S$, Hypothesis test $\mathcal{O}_k$ that takes qCFs as input and evaluates if the associated quarnet has a 4-blob or not (e.g., the cut test from [3]), Quadrapartition minimizer search algorithm $\mathcal{A}$ (e.g., 3f1a from [10])

**Output :** Hybrid block of $Part(v_{\mathcal{B}})$

**if** *fast mode* **then**
  $c \leftarrow 1$;
  Select one taxon arbitrarily from $P_i$ to use for all qCF calculations for all $i \in \{1, 2, \ldots, m_{\mathcal{B}}\}$
**else**
  $c \leftarrow m_{\mathcal{B}} - 3$
Build a function $F(x)$ that returns $i$ if taxon $x \in P_i$ for all $x \in \cup_{i=1}^{m_{\mathcal{B}}} P_i$;
Initialize an $m_{\mathcal{B}}$-vector, called *votes*, to zeros;
**for** $c$ *voting rounds* **do**
  Generate re-quadrapartition $P'_1|P'_2|P'_3|P'_4$ by randomly shuffling the blocks of $Part(v_{\mathcal{B}}) = P_1|P_2|\ldots|P_{m_{\mathcal{B}}}$ and then assigning $P_i$ to $P'_j$ if $i$ mod 4 equals $j$ for all $i \in \{1, 2, \ldots, m_{\mathcal{B}}\}$;
  $(p, \{x_1, x_2, x_3, x_4\}) \leftarrow \mathcal{A}(\{P'_1, P'_2, P'_3, P'_4\}, \mathcal{O}_k)$;
  Add one to $votes[F(x_1)]$, $votes[F(x_2)]$, $votes[F(x_3)]$, $votes[F(x_4)]$;
Let $P_a, P_b, P_c, P_d, P_e$ be the five blocks with the most votes, sorted from highest to lowest, breaking ties arbitrarily;
**if** *votes[a] > votes[b]* **then**
  **return** $P_a$;
**else**
  Apply $\mathcal{O}_k$ to all ways of choosing four blocks from $P_a, P_b, P_c, P_d, P_e$ and store the 4-block subset with highest $p$-value, denoted $P_s, P_t, P_y, P_z$;
  **return** $\{P_a, P_b, P_c, P_d, P_e\} \setminus \{P_s, P_t, P_y, P_z\}$;

▶ **Theorem 1.** *Assume the same settings as Lemma 1, and let $v_{\mathcal{B}}$ be some vertex in the TOB for $N^-$ with degree $m_{\mathcal{B}} \geq 5$. Then, Algorithm 1 with $c \geq 1$ voting rounds applied to $Part(v_{\mathcal{B}})$ and $k$ gene trees will, with probability $\to 1$ as $k \to \infty$, return the (correct) hybrid block of $Part(v_{\mathcal{B}})$. The time complexity of applying Algorithm 1 to all $v_{\mathcal{B}}$ is $O(n \log(n)k)$ in fast mode or $O(n^2 k)$ otherwise, where $n$ is the number of taxa.*

**Proof.** Vertex $v_{\mathcal{B}}$ is associated with a non-trivial blob $\mathcal{B}$ in $N^-$ given our assumption that $N^-$ is level-1 and binary, so the 3f1a algorithm applied to an arbitrary re-quadrapartition of $Part(v_{\mathcal{B}})$ is guaranteed to apply $\mathcal{O}_k$ to at least one 4-taxon subset $Y$ such that $N^-|_Y$ contains a 4-blob, implying that $Y$ contains a taxon from the hybrid block, by Theorem 2 in [10]. Then, by Lemma 1, the minimizer subset returned by 3f1a given an arbitrary re-quadrapartition of $Part(v_{\mathcal{B}})$ and $k$ gene trees will, with probability going to 1 as $k \to \infty$, include the hybrid block of $Part(v_{\mathcal{B}})$. Repeating for $c$ arbitrary re-quadrapartitions of $Part(v_{\mathcal{B}})$ will produce one of two outcomes: (1) the hybrid block is the only block that earns $c$ votes or (2) the hybrid block is tied for $c$ votes with one or more other blocks. Taking the five blocks with the most votes, breaking ties arbitrarily, ensures that exactly one way of selecting 4 out of these

5 blocks does not include the hybrid block. The $p$-value from applying the oracle $\mathcal{O}_k$ to this special 4-block subset will be higher than the other 4-block subsets by Lemma 1, revealing the hybrid block by exclusion. The argument above extends to *fast mode*, in which a single taxon is selected arbitrarily to represent each block of $Part(v_{\mathcal{B}})$ (Algorithm 1, line 3), as any 4-taxon subset that covers the same distinct blocks of $Part(v_{\mathcal{B}})$ yields the same expected qCFs under the NMSC given $\mathcal{N}^-$ [7] (also see [10]). It is worth noting that a 4-block tie occurs in fast mode because of setting $c = 1$. However, our implementation of NetCS uses $c = m_{\mathcal{B}} - 3$, as this does not increase the total time complexity of NetCS (Corollary 1).

Now we consider time complexity. Building the taxon-to-block mapping function $F$ is $O(m_{\mathcal{B}})$ in fast mode or $O(n)$ otherwise. Applying 3f1a to $c$ re-quadrapartitions requires $O(cm_{\mathcal{B}})$ or $O(cn)$ queries to the oracle $\mathcal{O}_k$, depending on whether fast mode is employed or not. The oracle is a hypothesis test that can be performed in constant time but requires qCFs as input. After an $O(n \log(n)k)$ preprocessing step, the well-known binary lifting algorithm can be used to retrieve the lowest common ancestor of any pair of leaves in an (arbitrarily) rooted binary gene tree in constant time [14], enabling qCF computation for any 4-taxon subset on the fly in $O(k)$ time. Therefore, when including the calculation of qCFs, each oracle query is $O(cm_{\mathcal{B}}k)$ in fast mode and $O(cnk)$ otherwise. The other calculations do not exceed these time costs. Repeating for all $v_{\mathcal{B}}$ gives $O(n \log(n)k + \sum_{v_{\mathcal{B}}} cm_{\mathcal{B}}k)$ in fast mode and $O(n \log(n)k + \sum_{v_{\mathcal{B}}} cnk)$ otherwise. Taking $c = 1$ in fast mode and $c = O(m_{\mathcal{B}})$ otherwise gives us our result, noting that $\sum_{v_{\mathcal{B}}} O(m_{\mathcal{B}}) = O(n)$. Note that hybrid detection requires qCFs to be computed for $O(n)$ 4-taxon subsets when running in fast mode; as expected qCFs reveal the quarnet topology, we say it makes $O(n)$ **quarnet queries**. ◀

## 3.2 Pivot Scanning

After determining the hybrid block of $Part(v_{\mathcal{B}})$, it remains to determine the circle order. NetCS leverages a *pivot* to linearize the circular order for sorting.

▶ **Definition 2** (Pivots). *Let $N^-$ be a binary, semi-directed, level-1 network. Then, any non-trivial blob $\mathcal{B}$ in $N^-$ has two reticulation edges whose source vertices are called* pivots.

The deletion of the cut edge incident to each pivot vertex induces a pivot block.

▶ **Lemma 2.** *Let $Part(\mathcal{B}) = P_1|\ldots|P_{m_{\mathcal{B}}}$ be a partition induced by a non-trivial blob $\mathcal{B}$ in a binary, semi-directed, level-1 network $N^-$, and let $P_h$ be the hybrid block of $Part(\mathcal{B})$. Then, $P_p$ is a pivot block of $Part(\mathcal{B})$ if and only if for all $s, t \in \{1, \ldots, m_{\mathcal{B}}\}$ such that $s \neq t \neq p \neq h$ and all $x_h \in P_h, x_p \in P_p, x_s \in P_s, x_t \in P_t$, $N^-|_{\{x_h,x_p,x_s,x_t\}}$ displays quartet $x_h, x_p|x_s, x_t$.*

The proof of Lemma 2 is in the Appendix A. The idea is that the taxa from the pivot and hybrid blocks must be siblings in one of the two quartets displayed by the quarnet. As the lowest of the three qCFs corresponds to the non-displayed quartet by Proposition 11 in [7], the qCFs identify one block, out of four, that is neither the hybrid nor one of the two pivots. Then, scanning through all blocks in $Part(v_{\mathcal{B}})$ gives the two pivot blocks by process of elimination (Algorithm 2 in Appendix A). This approach requires a method $\mathcal{M}_k$ for computing the qCFs based on four distinct blocks of $Part(v_{\mathcal{B}})$. In this setting, taxa from the same block are exchangeable, as previously noted, so in fast mode, one taxon is arbitrarily selected from each block in $\{P_h, P_p, P_s, P_t\}$ for qCF computation; however, computing the average qCFs from all 4-taxon subsets covering $\{P_h, P_p, P_s, P_t\}$ may increase robustness to noise, which occurs when qCFs are computed from a finite number of gene trees with estimation error and/or missing taxa. Both approaches enable consistent estimation.

▶ **Theorem 2.** *Assume the same settings as Lemma 1. Let $\pi$ be the error tolerance:*

$$\pi = \min_{Y \in f_4^4(S)} \min\{|\mathbb{P}_{\mathcal{N}^-}(q_1^Y) - \mathbb{P}_{\mathcal{N}^-}(q_3^Y)|, |\mathbb{P}_{\mathcal{N}^-}(q_2^Y) - \mathbb{P}_{\mathcal{N}^-}(q_3^Y)|\} \quad (1)$$

*where $f_4^4(S)$ is the set of all 4-taxon subsets $Y \subseteq S$ such that $N^-|_Y$ contains a 4-blob and $q_3^Y$ is the quartet out of the three possible on $Y$ that is not displayed by $N^-|_Y$. Let $v_{\mathcal{B}}$ be some vertex in the TOB for $N^-$ with degree $m_{\mathcal{B}} \geq 5$. Then, for every $\epsilon > 0$, Algorithm 2 applied to $Part(v_{\mathcal{B}})$, with hybrid block $P_h$, and $k \geq \frac{2}{\pi^2} \log \frac{6|f_4^4(S)|}{\epsilon}$ gene trees returns the (correct) two pivot blocks of $Part(v_{\mathcal{B}})$ with probability at least $1 - \epsilon$. The time complexity of applying Algorithm 2 to all $v_{\mathcal{B}}$ is $O(nk)$ in fast mode and $O(n^2k)$ otherwise.*

**Proof.** By Lemma 2, Algorithm 2 returns the two pivot blocks for all $v_{\mathcal{B}}$ when the lowest qCF on $Y$ corresponds to the non-displayed quartet for all $Y \in f_4^4(S)$ such that $N^-|_Y$ contains a 4-blob. In fast mode or otherwise, this occurs when the error in the observed qCFs is bounded by $\pi/2$, specifically: $w_k(q_i^Y)/k > \mathbb{P}_{\mathcal{N}^-}(q_i^Y) - \pi/2$ and $w_k(q_i^Y)/k < \mathbb{P}_{\mathcal{N}^-}(q_i^Y) + \pi/2$ for all $i = \{1, 2, 3\}$ and all $Y \in f_4^4(S)$. Following the approach in [33], let $X_g^{i,Y}$ be a random variable which takes on value 1 when the restriction of gene tree $G_g$ to $Y$ is isomorphic to $q_i^Y$ and takes on value 0 otherwise. Then, $(X_g^{i,Y})_{g=1}^k$ is a sequence of independent Bernoulli random variables with success probability $\mathbb{P}_{\mathcal{N}^-}(q_i^Y)$, and the observed qCF is $w_k(q_i^Y) = \sum_{g=1}^k X_g^{i,Y}$ with expected value $\mathbb{E}_{\mathcal{N}^-}[w_k(q_i^Y)] = k \cdot \mathbb{P}_{\mathcal{N}^-}(q_i^Y)$. Now, we can bound the probability that $w_k(q_i^Y)/k$ deviates from its expected value by $\pi/2$ or more using Hoeffding's inequality: $\mathbb{P}[|(w_k(q_i^Y) - \mathbb{E}_{\mathcal{N}^-}[w_k(q_i^Y)])/k| \geq \pi/2] \leq 2e^{-2k(\pi/2)^2} = 2e^{-k\pi^2/2}$ for all $i \in \{1, 2, 3\}$ and all $Y \in f_4^4(S)$. Let $E_{i,Y}$ denote the event that $|(w_k(q_i^Y) - \mathbb{E}_{\mathcal{N}^-}[w_k(q_i^Y)])/k| \geq \pi/2$. Then, by the union bound, $\mathbb{P}(\cup_{Y \in f_4^4(S)} \cup_{i=1}^3 E_{i,Y}) \leq \sum_{Y \in f_4(S)} \sum_{i=1}^3 \mathbb{P}(E_{i,Y}) \leq 6|f_4^4(S)|e^{-k\pi^2/2}$. Lastly, we solve $1 - 6|f_4^4(S)|e^{-k\pi^2/2} \leq 1 - \epsilon$ for $k$ to get our result. To summarize, Algorithm 2 is consistent under the NMSC and, as $|f_4^4(S)| = O(n^4)$, has sample complexity $O(\frac{1}{\pi^2} \log \frac{n}{\epsilon})$.

Now consider time complexity. Algorithm 2 computes qCFs for $O(m_{\mathcal{B}})$ 4-block subsets. Each qCF computation takes $O(k)$ time in fast mode, taking advantage of the preprocessing performed for hybrid detection; otherwise, computing averages via the algorithm in Fig. S14 of [32] takes $O(nk)$ time. Repeating for all $v_{\mathcal{B}}$ gives $O(\sum_{v_{\mathcal{B}}} m_{\mathcal{B}}k) = O(nk)$ and $O(\sum_{v_{\mathcal{B}}} m_{\mathcal{B}}nk) = O(n^2k)$. Pivot scan makes $O(n)$ quartet queries in fast mode. ◀

## 3.3 Circular Order Sorting

Taking the hybrid block $P_h$ and a pivot block $P_p$ together enables us to linearize the circular order of blob $\mathcal{B}$ as the canonical vector $V_{\mathcal{B}}(P_h, P_p) = [P_h, P_p, ..., P_{p'}]$, where $P_{p'}$ is the block of $Part(\mathcal{B})$ induced by the other pivot. Then, mapping each block $P_s \in Part(\mathcal{B})$ to its index in the canonical vector $V_{\mathcal{B}}(P_h, P_p)$ yields a total ordering on the blocks, denoted $L_{\mathcal{B}}(P_h, P_p)$. As an example, traversing the only cycle in $N^-$ of Fig. 1A from the hybrid vertex $u_1$ towards the pivot $u_2$, ignoring the direction of reticulation edges, gives $V_{\mathcal{B}}(P_1, P_2) = [P_1, P_2, P_3, P_4, P_5, P_6]$. Traversing in the opposite direction towards the other pivot $u_6$ gives $V_{\mathcal{B}}(P_1, P_6) = [P_1, P_6, P_5, P_4, P_3, P_2]$. Now consider the relation in $L_{\mathcal{B}}(P_h, P_p)$ for two distinct blocks $P_s, P_t \in Part(\mathcal{B})$ that are neither the hybrid $P_h$ nor the focal pivot $P_p$.

▶ **Lemma 3.** *Let $Part(\mathcal{B}) = P_1|\dots|P_{m_{\mathcal{B}}}$ be a partition induced by a non-trivial blob $\mathcal{B}$ in a binary, semi-directed, level-1 network $N^-$. Let $P_h$ be the hybrid block and $P_p$ be a pivot block of $Part(\mathcal{B})$. Then, for all pairs of blocks $P_s, P_t \in Part(\mathcal{B})$ such that $s \neq t \neq h \neq p$, $P_s < P_t$ is in $L_{\mathcal{B}}(P_h, P_p)$ if and only if for all $x_h \in P_h, x_p \in P_p, x_s \in P_s, x_t \in P_t$, $N^-|_{\{x_h, x_p, x_s, x_t\}}$ displays quartet $x_t, x_h|x_p, x_s$; otherwise, $P_t < P_s$ is in $L_{\mathcal{B}}(P_h, P_p)$.*

The proof of Lemma 3 is in the Appendix A. The idea is that one of the two displayed quartets will put the hybrid and the focal pivot together and the other will separate the hybrid and the focal pivot, revealing the linearized order of the non-pivot, non-hybrid blocks $P_s, P_t$. As the lowest of the three qCFs corresponds to the non-displayed quartet by Proposition 11 in [7], the ordering of $P_s, P_t$ can be determined by comparing the qCF values of the two quartets that separate the pivot and the hybrid (Algorithm 3 in Appendix A). This approach is closely related to the identifiability result for circular orders; however, the innovation behind NetCS is leveraging a pivot to linearize the order and then recovering it in an efficient manner via sorting (Algorithm 4 in Appendix A).

▶ **Theorem 3.** *Assume the same settings as Lemma 1. Let $v_{\mathcal{B}}$ be some vertex in the TOB for $N^-$ with degree $m_{\mathcal{B}} \geq 5$. Then, for every $\epsilon > 0$, Algorithm 4 applied to $Part(v_{\mathcal{B}})$, with hybrid block $P_h$ and pivot block $P_p$, and $k \geq \frac{2}{\pi^2} \log \frac{6|f_4^4(S)|}{\epsilon}$ gene trees returns the (correct) circular order of $Part(v_{\mathcal{B}})$ with probability at least $1-\epsilon$. The total time complexity of applying Algorithm 4 to all $v_{\mathcal{B}}$ is $O(n \log nk)$ in fast mode and $O(n^2 \log(n)k)$ otherwise, where $n$ is the number of taxa.*

**Proof.** By Lemma 3, Algorithm 3 returns the correct relation in $L_{\mathcal{B}}(P_h, P_p)$ for all pairs $P_s, P_t \in Part(v_{\mathcal{B}})$ such that $s \neq t \neq h \neq p$ provided the lowest qCF corresponds to the non-displayed quartet for all $Y \in f_4^4(S)$ such that $N^-|_Y$ contains a 4-blob. Following the same argument in the proof of Theorem 2, this occurs with probability at least $1-\epsilon$ when the number $k$ of gene trees is at least $\frac{2}{\pi^2} \log \frac{6|f_4^4(S)|}{\epsilon}$. And, when the correct relations are returned, the (correct) circular order of $Part(\mathcal{B})$ is returned by merge sort (Algorithm 3–4).

Now consider time complexity. Merge sort makes $O(m_{\mathcal{B}} \log m_{\mathcal{B}})$ pairwise block comparisons. Each comparison requires qCF computation, which is $O(k)$ time in fast mode or $O(nk)$ time otherwise, as previously discussed for pivot scan. Repeating for all $v_{\mathcal{B}}$ gives us our result. Circular sorting makes $O(n \log n)$ quarnet queries when running in fast mode. ◀

## 3.4 Theoretical and Practical Considerations

Combining Theorems 1, 2, and 3 gives our final result:

▶ **Corollary 1.** *NetCS is statistically consistent under the NMSC and has time complexity $O(n \log(n)k)$ in fast mode or $O(n^2 \log(n)k)$ otherwise for $k$ genes and $n$ species. In fast mode, NetCS computes qCFs for $O(n \log n)$ 4-taxon subsets, referred to as quarnet queries.*

NetCS's fast mode is for the regime where the error in all qCFs is below the tolerance threshold $\pi/2$, so every quarnet can be reconstructed with perfect accuracy (i.e., we have an oracle for quarnets). Only $O(n)$ quarnets are accessed for hybrid voting and pivot scan; this increases to $O(n \log n)$ quarnets for circular sorting; thus, NetCS matches the lower bound of quarnet complexity recently established for semi-directed, level-1 network reconstruction (Proposition 6 in [13]). Moreover, when given direct access to quarnets, rather than inferring them from qCFs, NetCS has time complexity $O(n \log n)$. This improves upon the time complexity of full semi-directed level-1 network reconstruction from quarnets, which can be can be done in $O(n^2)$ using the taxon-addition-like algorithm recently presented by Frohn *et al.* (2025) (Theorem 13 in [13]). Like fast mode, the results in [13] are primarily of theoretical interest given the assumption of a perfect oracle (and there is no implementation of their algorithm to our knowledge).

Although our analysis primarily focused on fast mode, we initially designed the slower version NetCS, with the goal of increasing robustness to noisy qCFs, and only later defined the fast version to enable comparisons to [13]. Beyond the extra computation previously

mentioned, our implementation of NetCS also runs circular sorting taking each block in $Part(v_{\mathcal{B}})$, excluding the hybrid, as a candidate pivot, with the goal of increasing robustness to noisy qCFs. to increase the robustness. An error boolean is returned for each pairwise block comparison made during sorting with Algorithm 3, indicating whether the quartet with the hybrid and candidate pivot as siblings had the lowest qCF. This scenario should never occur if the candidate is the correct pivot by Theorem 2. Thus, NetCS reconstructs the blob taking the circular order with the lowest total error score, breaking ties to favor the candidate pivots originally returned from pivot scan and otherwise breaking ties arbitrarily. These additional steps do not impact statistical consistency because both pivots returned by the pivot scan will have score 0, with probability going to 1 as the number of gene trees goes to infinity, at which point, no other (incorrect) candidate can achieve a lower total error score. However, testing all candidate pivot blocks increases the total time complexity of NetCS to $O(n^3 \log(n)k)$. As we will show, even with this additional computation, NetCS is still fast in practice likely because it avoids accessing qCFs for all subsets of four taxa. Precomputation of all qCFs not only has an $O(n^4k)$ time cost but also an $\Omega(n^4)$ storage cost, which can lead to large constant factors in qCF access from the memory hierarchy (e.g., cache misses during either qCF precomputation or blob reconstruction).

NetCS is not without practical limitations, including those inherited from employing the same algorithmic framework of NANUQ+. First, an estimated TOB is required as input, which may contain varying levels of error. Second, all blobs are reconstructed independently, which can lead to an incompatible set of inferred hybrids (i.e., no valid semi-directed network exists that realizes all inferred hybrids simultaneously [18]). To address this issue, our implementation of NetCS processes blobs in decreasing order of degree and excludes blocks that conflict with previously inferred hybrids during the voting procedure. If all blocks are excluded, NetCS does not attempt to reconstruct that blob and thus fails to reconstruct the network. Likewise, NANUQ+ can fail to produce a valid output network due to conflicts among inferred hybrids.

## 3.5 Experimental Evaluation

To study the empirical performance of NetCS, we conducted an evaluation study using data sets simulated under the NMSC, so the true level-1 network and its tree of blobs were known. Details, including software commands and version numbers, are given in Appendix B.

**Experiment 1.** The goal of experiment 1 was to evaluate the accuracy of level-1 blob reconstruction with NetCS. Data sets [20] used for this evaluation were previously simulated by [10] with 12 model conditions corresponding to three numbers of taxa (50, 100, and 200) and four branch length scaling factors (2.0, 1.0, 0.5, 0.25). The scaling factors varied the level of gene tree discordance due to ILS (note that the mean discordance levels were 45%, 65%, 80%, and 90%, respectively, based on simulated level-0 networks). Twenty replicate networks were available per model condition. For each replicate, 1000 gene trees were simulated under the NMSC with independent inheritance. The number of (true) gene trees given as input to NetCS was varied from 100, 500, and 1000, yielding 36 conditions in total.

NetCS was given the true TOB as input so that hybrid detection and circular order reconstruction could be evaluated for blobs with degree at least 5. **Hybrid error** was defined as the proportion of blobs for which the predicted hybrid block differed from the true hybrid block. **Circular order error** was quantified using the Kendall tau distance. For two linear permutations $\sigma_1, \sigma_2$ on the set $B = \{1, 2, \ldots, m_{\mathcal{B}}\}$, the Kendall tau distance $D_{k\tau}(\sigma_1, \sigma_2)$ is the fraction of unordered pairs $(i, j) \in B$ whose relative ordering differs between $\sigma_1$ and

$\sigma_2$ [9]. We extended this notion to circular orders by considering all linearizations obtained by cutting the circle at each position as well as its reversal. Let $\Sigma(\zeta)$ denote the set of all linearizations of circular order $\zeta$, and let $\Sigma(\zeta)^R$ denote their reversals. Then, the Kendall tau distance between two circular orders $\zeta_1, \zeta_2$ is

$$D_{k\tau}(\zeta_1, \zeta_2) = \min_{(\sigma_1,\sigma_2)\in(\Sigma(\zeta_1)\cup\Sigma(\zeta_1)^R)\times(\Sigma(\zeta_2)\cup\Sigma(\zeta_2)^R)} D_{k\tau}(\sigma_1, \sigma_2). \tag{2}$$

We reported hybrid error and circular order error averaged over all blobs of degree at least 5 in the true tree of blobs for each replicate network.

**Experiment 2.** The goal of experiment 2 was to benchmark NetCS against NANUQ+ in terms of blob reconstruction error as well as computational efficiency. This evaluation used the same data sets as experiment (1), except that true gene trees were replaced with gene tree estimated from simulated DNA sequences using IQ-TREE model selection [26, 21] (sometimes gene tree estimation from simulated DNA sequences failed but the total number of estimated gene trees was always greater than 900). Mean gene tree estimation error, quantified in terms of (normalized) Robinson-Foulds error [31], ranged from 40% to 70%. Both NetCS and NANUQ+ were given the true TOB as input so that reconstructed blobs could be compared. Computational efficiency was evaluated as wallclock time (24-hour limit) and peak resident set size (peak RSS; 256 GB limit) (note that the reported time and memory usage does not include TOB reconstruction, as the true TOB was given as input). Both methods were run on compute nodes with the same architecture (AMD EPYC 7313) when collecting these performance metrics. NANUQ+ was given access to 8 threads for qCF precomputation.

**Experiment 3.** The goal of the third experiment was to evaluate NetCS for end-to-end semi-directed level-1 network reconstruction in comparison with NANUQ+ and CAMUS. Data sets used for this evaluation were previously simulated for the study presenting CAMUS [39] (note that CAMUS could not be run in the previous experiments because the networks did not have a single taxon outgroup, which led to issues correctly rooting CAMUS' estimated base tree). The CAMUS data sets had four model conditions with 51, 101, 151, and 201 taxa, including a single taxon outgroup. Replicate networks with blobs of degree 4 were excluded, as the hybrid is not identifiable for those blobs; this left 47, 45, 43, and 49 replicates (out of 50) for the 51-, 101-, 151-, and 201-taxon conditions, respectively. Each replicate had 1000 gene trees. Mean gene tree estimation error was $\sim 20\%$; ILS level was not reported by [38].

Since experiment 3 was based on data sets from the study presenting CAMUS [38], CAMUS was run with the same default and recommended settings. The base tree for CAMUS was estimated with ASTRAL-IV [42] and rooted at the outgroup taxon. Unlike the other methods, CAMUS returns a *collection* of (binary) *directed*, level-1 networks, one after each edge addition; we used the directed network with the same number of edge additions as the number of non-trivial blobs (i.e., hybrid vertices) as the true network, with the goal of giving CAMUS an advantage. For running NetCS and NANUQ+, the ASTRAL-IV tree was used to estimate a TOB following the approach of [10]; see Appendix B for details. NetCS and NANUQ+ were run twice, once given the estimated TOB as input and once given the true TOB as input.

All methods were evaluated based on hybrid ancestry error and network topology error. A species has **hybrid ancestry** if it is the descendant of one or more hybrids. False positives (FPs) correspond to the estimated network predicting a species has hybrid ancestry when it does not according to the true network; false negatives (FNs) are the reverse. FPR is the number of FPs divided by the number of species with hybrid ancestry according to the

estimated network. FNR is the number of FNs divided by the number of species with hybrid ancestry according to the true network. Balanced error rate, defined as the arithmetic mean of the FPR and FNR, was reported for all methods. Network topology error was evaluated as the **normalized hardwired cluster distance (HWCD)** [19]. Each edge in a directed network induces a hardwired cluster of all the taxon descendants of the edge. The hardwired clusters induced by all edges of the network form the hardwired cluster system. The HWCD is the symmetric difference of the hardware cluster systems of two directed networks. When comparing level-1 network, HWCD is a distance [25, 6, 8]. The extension to semi-directed networks takes the minimum value from all possible ways to "root" the two networks [18]. The normalized version divides HWCD by the total number of edges in both networks minus the total number of leaves in both networks. We used the implementation of HWCD in PhyloNetworks [35], treating all networks as semi-directed.

## 4 Experimental Results

We now present the results of the three experiments.

**Experiment 1.** The goal of experiment 1 was to evaluate how model condition impacted level-1 blob reconstruction with NetCS (Table 2, Fig. 2). As the number of gene trees and the branch scale factor increased, error decreased. For example, the average Kendall tau distance dropped from 0.15 to 0.0 going from the hardest model condition (0.25×, 100 true gene trees) to the easiest (2.0×, 1000 true gene trees), both with 50 taxa. These trends are expected as larger numbers of true gene trees correspond to lower error in observed qCFs and larger branch length scaling factors correspond to lower ILS, which, in turn, corresponds to higher error tolerance $\pi$. Similar trends were observed for model conditions with larger numbers of taxa and hybrid error. Average hybrid error was generally low at 1000 gene trees, ranging from 0.123 to 0.000 for the easiest to hardest 50-taxon model conditions (see Table 2 for details). NetCS did not have any failures in experiment 1.

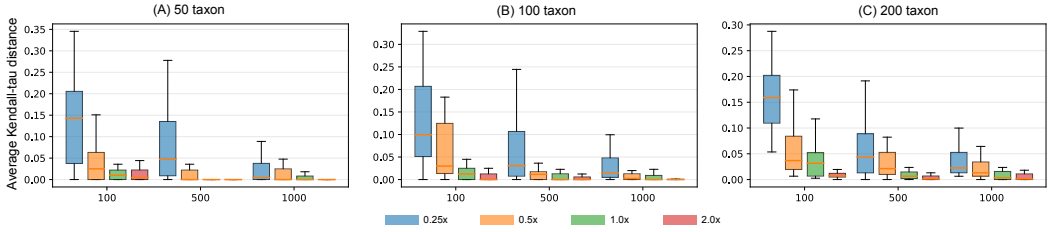



**Figure 2 Experiment 1: Average Kendall-tau distance**. Three subfigures (A), (B),(C) correspond to data sets with 50 taxa, 100 taxa, and 200 taxa, respectively. The $x$-axis is the number of gene trees. Each colored box represents a different branch scaling factor in order of decreasing ILS. The $y$-axis is error in circular order reconstruction (i.e., Kendall-tau distance, averaged across all blobs in replicate). Orange lines represent medians. Outliers are not shown.

**Experiment 2.** The goal of experiment 2 was to compare NetCS against NANUQ+ in terms of level-1 blob reconstruction error and computational efficiency (Table 1, Fig. 6). Again, the percent hybrids correctly predicted increased with decreasing ILS level, with NetCS making the correct call for 79%, 83%, 88%, 95% of blobs in 50-taxon data sets, respectively. Net-CS outperformed NANUQ+ in hybrid detection on 5 model conditions, the reverse occurred on 1 model condition, and the two methods tied in the remaining 2 model conditions.

Error in circular order reconstruction also decreased with increasing ILS level, with NetCS and NANUQ+ tied, each outperforming the other on 4 conditions. The difference between methods for both tasks was small, suggesting they achieved similar accuracy.

In contrast, NetCS and NANUQ+ differed substantially in terms of their computational efficiency. On the 50-taxon data sets, Net-CS completed in under 5 seconds on average while using less than 0.33GB memory, whereas NANUQ$^+$ required ∼7–8 minutes on average and over 0.7GB memory. The performance gap increased for the 100-taxon data sets. Net-CS completed in 20 seconds on average while using less than 0.4GB memory, whereas NANUQ$^+$ required more than one hour on average and over 50GB memory. On the 200-taxon data sets, NANUQ$^+$ either exceeded the 24-hour wall-clock limit or the 256GB memory limit, while NetCS completed within 15–215 seconds using only about 0.5GB memory. Note that NetCS did not have any failures for experiment 2, but NANUQ+ failed to return a valid network for one 50-taxon network (replicate #1207) for two branch scaling factors (0.25× and 0.5×); these two replicate data sets were excluded from error and runtime comparisons.

**Table 1 Experiment 2.** The table shows the number of blobs, across all replicate networks for the same model condition, in which NetCS is better / worse / tied with NANUQ+ in terms of hybrid error and circular order error (Kendall tau distance). The Kendall-tau distance is the average across all blobs in a replicate, which is then averaged across all replicates. Runtime and peak RSS values are averages across all replicates per model condition. Results are not shown for 200 taxa, as NANUQ+ did not complete within the maximum wallclock time of 24 hours or exceeded the memory limit of 256GB. Both methods were given up to 1000 estimated gene trees.

| # of | ILS | # of blob hybrid error | | | # of blob Kendall-tau | | | # of correct hybrids | | Average Kendall-tau | | Runtime (MB) | | Peak RSS (secs) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| taxa | Level | Better | Worse | Tie | Better | Worse | Tie | Net-CS | NANUQ+ | Net-CS | NANUQ+ | Net-CS | NANUQ+ | Net-CS | NANUQ+ |
| 50 | 0.25× | 3 | 3 | 18 | 6 | **9** | 9 | 19 | 19 | **0.043** | 0.048 | **330** | 709 | **5** | 480 |
| 50 | 0.5× | 1 | **2** | 21 | **8** | 3 | 13 | 20 | **21** | 0.018 | **0.012** | **330** | 710 | **5** | 466 |
| 50 | 1.0× | **2** | 1 | 24 | 5 | **6** | 16 | **24** | 23 | 0.020 | **0.011** | **329** | 709 | **4** | 444 |
| 50 | 2.0× | **1** | 0 | 23 | **5** | 3 | 16 | **23** | 22 | **0.011** | 0.013 | **329** | 708 | **4** | 440 |
| 100 | 0.25× | 1 | 1 | 28 | 8 | **9** | 13 | 27 | 27 | 0.034 | **0.031** | **396** | 50582 | **20** | 4711 |
| 100 | 0.5× | **2** | 1 | 30 | 9 | **11** | 13 | **31** | 30 | **0.022** | 0.034 | **396** | 50183 | **17** | 4415 |
| 100 | 1.0× | **2** | 0 | 31 | **10** | 9 | 14 | **32** | 30 | **0.016** | 0.018 | **395** | 50741 | **12** | 4158 |
| 100 | 2.0× | **1** | 0 | 32 | **9** | 7 | 17 | **32** | 31 | **0.010** | 0.026 | **392** | 51384 | **10** | 4230 |

**Experiment 3.** The goal of the final experiment was to benchmark NetCS as part of an end-to-end pipeline for network reconstruction against NANUQ+ and CAMUS. In terms of hybrid ancestry error, all methods exhibited poor performance (Fig. 3). CAMUS had the highest variability, sometimes achieving good predictions (balanced error close to 0) and other times not (balanced error close to 1). NANUQ+ and NetCS predicted hybrid ancestry at no better than a random guess (balanced error close to 0.5) and sometimes much worse. The poor performance of NANUQ+ and NetCS was due to high FPR and low FNR (i.e., many species were predicted to be hybrid descendants when they were not according to the true network). Manual inspection of estimated TOBs revealed under-refinement compared to true TOBs, especially towards the root. As a consequence, NetCS and NANUQ+ reconstructed more blobs and/or larger blobs compared to the true network, especially towards the root, leading to increased hybrid ancestry predictions. Providing NetCS and NANUQ+ with the true TOB substantially improved accuracy, dropping the balanced error rate to near 0 for all model conditions.

For network topology error, the best end-to-end method was CAMUS, with HWCD between 0.10–0.15 for all model conditions (Fig. 4). In contrast, HWCD was between 0.65–0.75 for NetCS and NANUQ+. This difference in performance is again explained by poor quality TOBs, as providing NetCS and NANUQ+ with the true TOBs dropped HWCD to near 0. Incidentally, we found that the good performance of CAMUS could be attributed

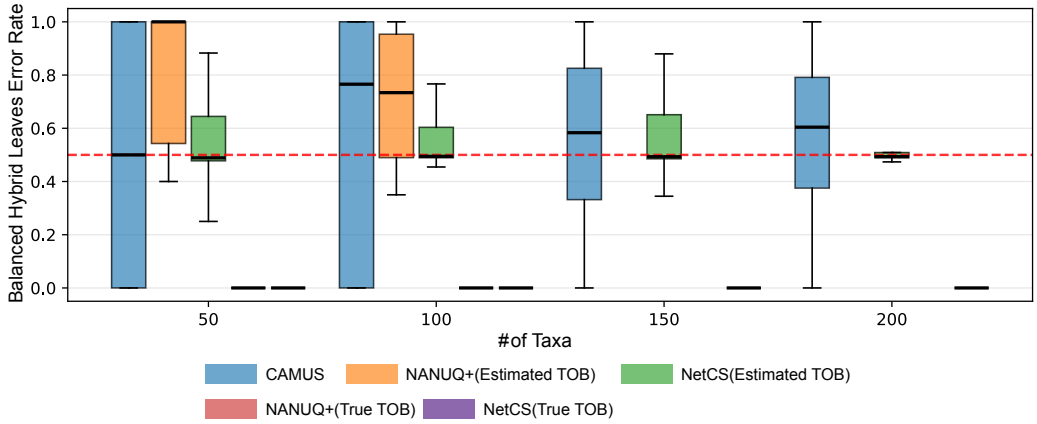



**Figure 3 Experiment 3: Hybrid ancestry balanced error rate.** The $y$-axis is the hybrid ancestry balanced error rate. The $x$-axis is the number of taxa. Each box represents a method, from left to right: CAMUS, NANUQ+ (given estimated TOB), NetCS (given estimated TOB), NANUQ+ (given the true TOB), and NetCS (given the true TOB). Orange lines represent medians. Outliers are not shown. Red dashed line represents the error rate of a random guess.

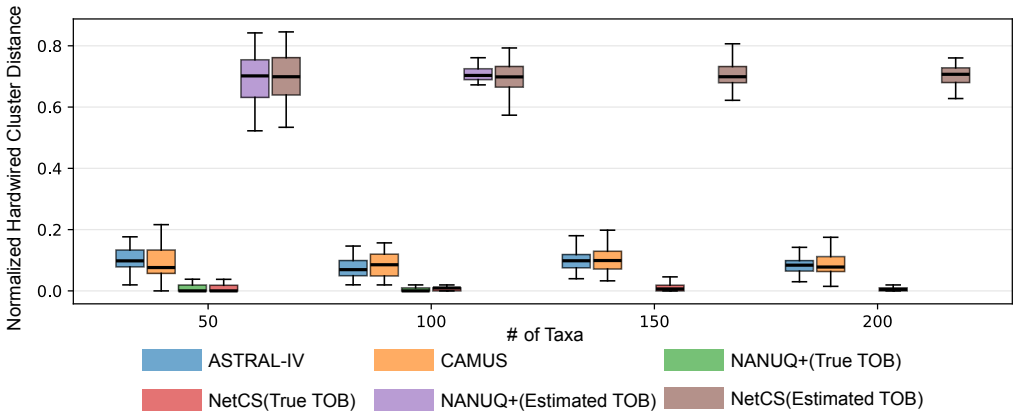



**Figure 4 Experiment 3: Normalized HWCD.** The $y$-axis is network topological error (normalized HCWD). The $x$-axis is the number of taxa. Each box represents a method, from left to right: ASTRAL-IV, CAMUS, NANUQ+ (given the true TOB), NetCS (given the true TOB), NANUQ+ (given estimated TOB), and NetCS (given estimated TOB). Orange lines represent medians. Outliers are not shown.

to the ASTRAL-IV base tree. Even before reticulation edges were added by CAMUS, the HWCD of the ASTRAL-IV tree was very close to the network produced by CAMUS. Moreover, the HWCD of the CAMUS network was better/worse than its ASTRAL-IV base tree in 27/20, 20/25, 23/20, and 27/22 replicates for the 51-, 101-, 151-, and 201-taxon model conditions. Thus, edge additions made by CAMUS increased error almost as often as they decreased error.

No results are shown for NANUQ$^+$ on 150- and 200-taxon data sets, as it did not complete within the maximum wallclock time of 24 hours or exceeded the memory limit of 256GB. NetCS (given an estimated TOB) failed to return a valid network for 2 replicates with 100 taxa, 2 with 150 taxa, and 8 with 200 taxa. These replicates were not excluded from the results presented for other methods.

## 5 Conclusions

In this paper, we studied the problem of reconstructing level-1 blobs under the NMSC model when the tree of blobs is known. This work was motivated by recent advances in tree of blobs reconstruction [3, 17, 13, 10, 30] and by the two-step design of NANUQ+ for reconstructing semi-directed level-1 networks. Our method, NetCS, combines majority voting to identify the hybrid vertex with sorting to recover the circular order. These primitives closely reflect prior identifiability results for semi-directed level-1 networks [34, 7, 1], although, to our knowledge, they had not previously been operationalized in a fast, statistically consistent method.

A key contribution of NetCS is that it avoids the precomputation of all qCFs by (1) repurposing the 3f1a algorithm from our recent study [10] and (2) applying an efficient sorting algorithm. This gives NetCS time complexity $O(n \log(n)k)$ and only $O(n \log n)$ quarnet queries, matching the lower bound of [13]. These results indicate that consistent reconstruction of level-1 blobs under the NMSC is easy, at least from a theoretical perspective.

Notably, our implementation of NetCS performs additional computation to improve robustness to noisy qCF estimates, leading to $O(n^3 \log(n)k)$ time complexity. However, by avoiding the quartic cost of qCF precomputation, NetCS enables analyses of data sets with 200 taxa and 1000 genes in only a few minutes, substantially improving upon NANUQ+ in both runtime and memory usage. Despite its reduced access to qCFs, NetCS was still highly competitive with NANUQ+ in terms of accuracy, achieving comparable error rates for both hybrid detection and circular order reconstruction when provided with the true tree of blobs. However, when supplied with an estimated, and poor quality, tree of blobs, the accuracy of both methods deteriorated substantially. This finding suggests that the primary practical challenge may lie in fast and accurate tree of blobs reconstruction.

Direct network reconstruction circumvents this issue, motivating us to benchmark NetCS and NANUQ+ against the recently proposed method CAMUS. We did not evaluate SNaQ because we used the same data sets as the CAMUS study, which had already shown that SNaQ was less topologically accurate than CAMUS. Like SNaQ, CAMUS outperformed NetCS and NANUQ+ in terms of network topology error; however, the opposite trend was observed when the true tree of blobs was available, rather than an estimated, and poor quality, tree of blobs. Moreover, CAMUS struggled to reliably determine whether species evolved through hybridization, a question of central importance in many biological studies. We also found that the topological accuracy of CAMUS was strongly correlated with that of the ASTRAL-IV base tree and that its edge addition procedure degraded topological accuracy nearly as often as it improved it.

Collectively, these findings highlight the need for improved methods for level-1 network reconstruction. Given the computational efficiency and strong performance of NetCS when provided with the true tree of blobs, a promising direction for future research is to combine NetCS with improved methods for tree of blobs reconstruction. The algorithmic techniques underlying NetCS also could be explored in the context of direct network reconstruction or other approaches for level-1 blob reconstruction. For example, reducing the number of 4-taxon subsets sampled in NANUQ+ or replacing collections of 4-taxon subsets with averages. Such techniques could also be relevant for applications of the recent method Squirrel [16] to gene tree data. Squirrel is similar to NANUQ+, in that both are based on quartet distances, but was introduced in the context of level-1 blob reconstruction from DNA sequences.

## References


1 Elizabeth S Allman, Hector Baños, Marina Garrote-Lopez, and John A Rhodes. Identifiability of level-1 species networks from gene tree quartets. *Bulletin of Mathematical Biology*, 86(9):110, 2024. doi:10.1007/s11538-024-01339-4.

2 Elizabeth S Allman, Hector Baños, Jonathan D Mitchell, and John A Rhodes. The tree of blobs of a species network: identifiability under the coalescent. *Journal of Mathematical Biology*, 86(1):10, 2023. doi:10.1007/s00285-022-01838-9.

3 Elizabeth S Allman, Hector Baños, Jonathan D Mitchell, and John A Rhodes. TINNiK: inference of the tree of blobs of a species network under the coalescent model. *Algorithms for Molecular Biology*, 19(1):23, 2024. doi:10.1186/s13015-024-00266-2.

4 Elizabeth S Allman, Hector Baños, John A Rhodes, and Kristina Wicke. NANUQ+: a divide-and-conquer approach to network estimation. *Algorithms for Molecular Biology*, 20(1):14, 2025. doi:10.1186/s13015-025-00274-w.

5 Cécile Ané, John Fogg, Elizabeth S Allman, Hector Baños, and John A Rhodes. Anomalous networks under the multispecies coalescent: theory and prevalence. *Journal of Mathematical Biology*, 88(3):29, 2024. doi:10.1007/s00285-024-02050-7.

6 Allan Bai, Péter L Erdős, Charles Semple, and Mike Steel. Defining phylogenetic networks using ancestral profiles. *Mathematical Biosciences*, 332:108537, 2021.

7 Hector Baños. Identifying species network features from gene tree quartets under the coalescent model. *Bulletin of Mathematical Biology*, 81(2):494–534, 2019. doi:10.1007/s11538-018-0485-4.

8 Gabriel Cardona, Joan Carles Pons, Gerard Ribas, and Tomás Martínez Coronado. Comparison of orchard networks using their extended $\mu$-representation. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 21(3):501–507, 2024. doi:10.1109/TCBB.2024.3361390.

9 Vincent A Cicirello. Kendall tau sequence distance: Extending Kendall tau from ranks to sequences. *arXiv preprint arXiv:1905.02752*, 2019. arXiv:1905.02752.

10 Junyan Dai, Yunheng Han, and Erin K Molloy. Quartet-based species tree methods enable fast and consistent tree of blobs reconstruction under the network multispecies coalescent. *bioRxiv*, 2025. doi:10.1101/2025.11.05.686850.

11 Vu Dinh and Hector Baños. Misspecification strikes: Astral can mislead in the presence of hybridization, even for nonanomalous scenarios. *Molecular Biology and Evolution*, 42(3):msaf049, 2025. doi:10.1093/molbev/msaf049.

12 John Fogg, Elizabeth S Allman, and Cécile Ané. PhyloCoalSimulations: A simulator for network multispecies coalescent models, including a new extension for the inheritance of gene flow. *Systematic Biology*, 72(5):1171–1179, 2023. doi:10.1093/sysbio/syad030.

13 Martin Frohn, Niels Holtgrefe, Leo van Iersel, Mark Jones, and Steven Kelk. Reconstructing semi-directed level-1 networks using few quarnets. *Journal of Computer and System Sciences*, 152:103655, 2025. doi:10.1016/j.jcss.2025.103655.

14 D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, Cambridge, United Kingdom, 1997.

15 Dan Gusfield, Vikas Bansal, Vineet Bafna, and Yun S. Song. A decomposition theory for phylogenetic networks and incompatible characters. *Journal of Computational Biology*, 14(10):1247–1272, 2007. doi:10.1089/cmb.2006.0137.

16 Niels Holtgrefe, Katharina T Huber, Leo van Iersel, Mark Jones, Samuel Martin, and Vincent Moulton. Squirrel: Reconstructing semi-directed phylogenetic level-1 networks from four-leaved networks or sequence alignments. *Molecular Biology and Evolution*, 42(4):msaf067, 2025. doi:10.1093/molbev/msaf067.

17 Katharina T Huber, Leo van Iersel, Mark Jones, Vincent Moulton, and Leonie Veenema-Nipius. When are quarnets sufficient to reconstruct semi-directed phylogenetic networks? *Bulletin of Mathematical Biology*, 87(10):136, 2025. doi:10.1007/s11538-025-01510-5.

**18** Katharina T. Huber, Leo van Iersel, Remie Janssen, Mark Jones, Vincent Moulton, Yukihiro Murakami, and Charles Semple. Orienting undirected phylogenetic networks. *Journal of Computer and System Sciences*, 140:103480, 2024. doi:10.1016/j.jcss.2023.103480.

**19** Daniel H Huson, Regula Rupp, and Celine Scornavacca. *Phylogenetic Networks*. Cambridge University Press, Cambridge, England, 2011.

**20** Erin K. Molloy Junyan Dai, Yunheng Han. Data from: Quartet-based species tree methods enable fast and consistent tree of blobs reconstruction under the network multispecies coalescent, 2026. doi:10.5061/dryad.cjsxksnnp.

**21** Subha Kalyaanamoorthy, Bui Quang Minh, Thomas KF Wong, Arndt Von Haeseler, and Lars S Jermiin. Modelfinder: fast model selection for accurate phylogenetic estimates. *Nature methods*, 14(6):587–589, 2017.

**22** Nathan Kolbow, Sungsik Kong, Tyler Chafin, Joshua Justison, Cécile Ané, and Claudia Solís-Lemus. SNaQ.jl: Improved scalability for phylogenetic network inference. *bioRxiv*, 2025. doi:10.1101/2025.11.17.688917.

**23** Nathan Kolbow, Sungsik Kong, and Claudia Solís-Lemus. A method for massively scalable phylogenetic network inference. *bioRxiv*, 2025. doi:10.1101/2025.05.05.652278.

**24** Sungsik Kong, David L Swofford, and Laura S Kubatko. Inference of phylogenetic networks from sequence data using composite likelihood. *Systematic Biology*, 74(1):53–69, 2024. doi:10.1093/sysbio/syae054.

**25** Michael Maxfield, Jingcheng Xu, and Cécile Ané. A dissimilarity measure for semidirected networks. *IEEE Transactions on Computational Biology and Bioinformatics*, 2025. doi:10.1109/TCBBIO.2025.3534780.

**26** Bui Quang Minh, Heiko A Schmidt, Olga Chernomor, Dominik Schrempf, Michael D Woodhams, Arndt von Haeseler, and Robert Lanfear. IQ-TREE 2: New Models and Efficient Methods for Phylogenetic Inference in the Genomic Era. *Molecular Biology and Evolution*, 37(5):1530–1534, 2020. doi:10.1093/molbev/msaa015.

**27** Erin K Molloy, Arun Durvasula, and Sriram Sankararaman. Advancing admixture graph estimation via maximum likelihood network orientation. *Bioinformatics*, 37(Supplement_1):i142–i150, 2021. doi:10.1093/bioinformatics/btab267.

**28** R Alexander Pyron, Kyle A O'Connell, Edward A Myers, David A Beamer, and Hector Baños. Complex hybridization in a clade of polytypic salamanders (plethodontidae: Desmognathus) uncovered by estimating higher-level phylogenetic networks. *Systematic Biology*, 74(1):124–140, 2025. doi:10.1093/sysbio/syae060.

**29** Bruce Rannala and Ziheng Yang. Bayes estimation of species divergence times and ancestral population sizes using DNA sequences from multiple loci. *Genetics*, 164(4):1645–1656, 2003. doi:10.1093/genetics/164.4.1645.

**30** John A. Rhodes, Elizabeth S. Allman, Cecile Ané, and Hector Baños. Statistical inference of the tree of blobs of a phylogenetic network from quartet concordance factors. *bioRxiv*, 2026. doi:10.64898/2026.05.28.728501.

**31** D.F. Robinson and L.R. Foulds. Comparison of phylogenetic trees. *Mathematical Biosciences*, 53(1):131–147, 1981. doi:10.1016/0025-5564(81)90043-2.

**32** Erfan Sayyari and Siavash Mirarab. Fast coalescent-based computation of local branch support from quartet frequencies. *Molecular Biology and Evolution*, 33(7):1654–1668, 2016. doi:10.1093/molbev/msw079.

**33** Shubhanshu Shekhar, Sebastien Roch, and Siavash Mirarab. Species tree estimation using ASTRAL: how many genes are enough? *IEEE/ACM transactions on computational biology and bioinformatics*, 15(5):1738–1747, 2017. doi:10.1109/tcbb.2017.2757930.

**34** Claudia Solís-Lemus and Cécile Ané. Inferring phylogenetic networks with maximum pseudolikelihood under incomplete lineage sorting. *PLOS Genetics*, 12(3):1–21, 2016. doi:10.1371/journal.pgen.1005896.

35 Claudia Solís-Lemus, Paul Bastide, and Cécile Ané. Phylonetworks: A package for phylogenetic networks. *Molecular Biology and Evolution*, 34(12):3292–3298, 2017. doi:10.1093/molbev/msx235.

36 Alexandros Stamatakis. RAxML version 8: a tool for phylogenetic analysis and post-analysis of large phylogenies. *Bioinformatics*, 30(9):1312–1313, 2014. doi:10.1093/bioinformatics/btu033.

37 Mike Steel. *Phylogeny: Discrete and random processes in evolution*, volume 89 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. Society for Industrial and Applied Mathematics, 2016.

38 James Willson and Tandy Warnow. CAMUS: Scalable phylogenetic network estimation. *bioRxiv*, 2026. doi:10.64898/2026.02.01.703143.

39 James Willson and Tandy Warnow. Dataset for CAMUS: Scalable phylogenetic network estimation, 2026. doi:10.13012/B2IDB-6892704_V1.

40 Yun Yu, James H. Degnan, and Luay Nakhleh. The probability of a gene tree topology within a phylogenetic network with applications to hybridization detection. *PLOS Genetics*, 8(4):1–10, 2012. doi:10.1371/journal.pgen.1002660.

41 Yun Yu, Jianrong Dong, Kevin J. Liu, and Luay Nakhleh. Maximum likelihood inference of reticulate evolutionary histories. *Proceedings of the National Academy of Sciences*, 111(46):16448–16453, 2014. doi:10.1073/pnas.1407950111.

42 Chao Zhang and Siavash Mirarab. Weighting by gene tree uncertainty improves accuracy of quartet-based species trees. *Molecular Biology and Evolution*, 39(12):msac215, 2022. doi:10.1093/molbev/msac215.

43 Chao Zhang, Rasmus Nielsen, and Siavash Mirarab. Aster: A package for large-scale phylogenomic reconstructions. *Molecular Biology and Evolution*, page msaf172, 2025. doi:10.1093/molbev/msaf172.


## A Supplemental Methods

▶ **Lemma 2.** *Let $Part(\mathcal{B}) = P_1|\dots|P_{m_\mathcal{B}}$ be a partition induced by a non-trivial blob $\mathcal{B}$ in a binary, semi-directed, level-1 network $N^-$, and let $P_h$ be the hybrid block of $Part(\mathcal{B})$. Then, $P_p$ is a pivot block of $Part(\mathcal{B})$ if and only if for all $s,t \in \{1,\dots,m_\mathcal{B}\}$ such that $s \neq t \neq p \neq h$ and all $x_h \in P_h, x_p \in P_p, x_s \in P_s, x_t \in P_t$, $N^-|_{\{x_h,x_p,x_s,x_t\}}$ displays quartet $x_h, x_p|x_s, x_t$.*

**Proof.** Because $x_h, x_p, x_s, x_t$ are drawn from distinct blocks of $Part(\mathcal{B})$ and $x_h$, in particular, is drawn from the hybrid block, $N^-|_{\{x_h,x_p,x_s,x_t\}}$ contains a 4-blob by Lemma 5 in [10]. Let $u_j$ denote the vertex in $\mathcal{B}$ such that the deletion of the one cut edge incident to $u_j$ induces the bipartition: $P_j|S \setminus P_j$ for all $j \in \{1,\dots,m_\mathcal{B}\}$.

$\Rightarrow$ Assume that $P_p$ is a pivot block of $Part(\mathcal{B})$. By definition, there exists a reticulation edge $(u_p, u_h)$ in $N^-$. After deleting the partner reticulation edge of $(u_p, u_h)$, any path of distinct edges between $x_h \in P_h$ and $x_p \in P_p$ in $N^-|_{\{x_h,x_p,x_s,x_t\}}$ contains edge $(u_p, u_h)$ but no other edges in $\mathcal{B}$. In contrast, any path of distinct edges between $x_h$ and $x_s \in P_s$ or $x_t \in P_t$ in $N^-|_{\{x_h,x_p,x_s,x_t\}}$ contains edge $(u_p, u_h)$ as well as the (non-reticulation) edge in $\mathcal{B}$ that is incident to $(u_p, u_h)$, which induces bipartition $x_h, x_p|x_s, x_t$ giving us our result.

$\Leftarrow$ For the sake of obtaining a contradiction, assume the condition holds but $P_p$ is not a pivot block. Then, there exists two pivot blocks $P_v, P_w$ where $w \neq v \neq p \neq h$. For all $x_h \in P_h, x_p \in P_p, x_v \in P_v, x_w \in P_w$, $N^-|_{\{x_h,x_p,x_v,x_w\}}$ contains a 4-blob that displays two quartets: $x_h, x_v|x_w, x_p$ from deleting reticulation edge $(u_w, u_h)$ and $x_h, x_w|x_v, x_p$ from deleting reticulation edge $(u_v, u_h)$, as described in the forward direction of the proof. As $N^-|_{\{x_h,x_p,x_s,x_t\}}$ displays two quartets given our assumption that $N^-$ is level-1 and binary [7], quartet $x_h, x_p|x_v, x_w$ is not displayed, which is a contradiction. ◀

**Algorithm 2** Pivot Scan Algorithm.

**input :** Partition of species set $S$ induced by vertex $v_{\mathcal{B}}$, denoted $Part(v_{\mathcal{B}}) = P_1|P_2|\ldots|P_{m_{\mathcal{B}}}$, Hybrid block of $Part(v_{\mathcal{B}})$, denoted $P_h$, Set of $k$ unrooted, binary gene trees each on (a subset of) $S$

**output :** Two pivot blocks of $Part(v_{\mathcal{B}})$

**if** *fast mode* **then**
  Select one taxon arbitrarily from $P_i$ to use for all qCF calculations, repeating for all $i \in \{1, 2, \ldots, m_{\mathcal{B}}\}$;
  $\mathcal{M}_k \leftarrow$ algorithm for computing qCF on fly based on binary lifting algorithm
**else**
  $\mathcal{M}_k \leftarrow$ algorithm for computing average qCF from Fig. S14 in [32]
$\mathcal{K} \leftarrow \{P_1, P_2, ..., P_m\} \setminus \{P_h\}$;
$P_s, P_t \leftarrow$ 2 arbitrary elements of $\mathcal{K}$; $\mathcal{K} \leftarrow \mathcal{K} \setminus \{P_s, P_t\}$;
**repeat**
  $P_v \leftarrow$ an arbitrary element in $\mathcal{K}$; $\mathcal{K} \leftarrow \mathcal{K} \setminus \{P_v\}$;
  $q_v \leftarrow$ qCF for $P_h, P_v|P_s, P_t$ computed via $\mathcal{M}_k$;
  $q_s \leftarrow$ qCF for $P_h, P_s|P_v, P_t$ computed via $\mathcal{M}_k$;
  $q_t \leftarrow$ qCF for $P_h, P_t|P_s, P_v$ computed via $\mathcal{M}_k$;
  **if** $q_s < q_v$ *and* $q_s < q_t$ **then** $P_s \leftarrow P_v$;
  **else if** $q_t < q_v$ *and* $q_t < q_s$ **then** $P_t \leftarrow P_v$;
**until** $\mathcal{K} = \emptyset$;
**return** $P_s, P_t$;

▶ **Lemma 3.** *Let $Part(\mathcal{B}) = P_1|\ldots|P_{m_{\mathcal{B}}}$ be a partition induced by a non-trivial blob $\mathcal{B}$ in a binary, semi-directed, level-1 network $N^-$. Let $P_h$ be the hybrid block and $P_p$ be a pivot block of $Part(\mathcal{B})$. Then, for all pairs of blocks $P_s, P_t \in Part(\mathcal{B})$ such that $s \neq t \neq h \neq p$, $P_s < P_t$ is in $L_{\mathcal{B}}(P_h, P_p)$ if and only if for all $x_h \in P_h, x_p \in P_p, x_s \in P_s, x_t \in P_t$, $N^-|_{\{x_h,x_p,x_s,x_t\}}$ displays quartet $x_t, x_h|x_p, x_s$; otherwise, $P_t < P_s$ is in $L_{\mathcal{B}}(P_h, P_p)$.*

**Proof.** Because $x_h, x_p, x_s, x_t$ are all drawn from distinct blocks of $Part(\mathcal{B})$ and $x_h$, in particular, is drawn from the hybrid block, $N^-|_{\{x_h,x_p,x_s,x_t\}}$ contains a 4-blob by Lemma 5 in [10]. Let $u_j$ denote the vertex in $\mathcal{B}$ such that the deletion of the one cut edge incident to $u_j$ induces the bipartition: $P_j|S \setminus P_j$ for all $j \in \{1, \ldots, m_{\mathcal{B}}\}$.

$\Rightarrow$ Assume $P_s < P_t$ is in $L_{\mathcal{B}}(P_h, P_p)$. Then, traversing $\mathcal{B}$ from the hybrid vertex $u_h$ toward pivot $u_p$ encounters $u_s$ before $u_t$. It follows that the restriction of $N^-$ to $\{x_h, x_p, x_s, x_t\}$ has two reticulation edges $(u_p, u_h)$ and $(u_t, u_h)$, and deleting reticulation edge $(u_p, u_h)$ gives the displayed quartet $x_t, x_h|x_p, x_s$.

$\Leftarrow$ For the sake of obtaining a contradiction, assume the condition holds but $P_t < P_s$ is in $L_{\mathcal{B}}(P_h, P_p)$. The restriction of $N^-$ to $\{x_h, x_p, x_s, x_t\}$ displays quartets $x_s, x_h|x_p, x_t$ and $x_h, x_p|x_s, x_t$ by the forward direction above and by Lemma 2, respectively. Only two quartets can be displayed given our assumption that $N^-$ is level-1 and binary [7]; thus, quartet $x_t, x_h|x_p, x_s$ is not displayed, which is a contradiction. ◀

**Algorithm 3** Linearized Circular Order Comparison Algorithm.

**input :** Distinct blocks $P_h, P_p, P_s, P_t$ from the partition of species set $S$ induced by vertex $v_{\mathcal{B}}$, where $P_h$ is the hybrid block and $P_p$ is the pivot block, Set of $k$ unrooted, binary gene trees each on (a subset of) $S$, A method $\mathcal{M}_k$ for computing qCFs from gene trees given 4 pairwise disjoint subsets of $S$

**output :** Relation of $P_s$ and $P_t$ in $L_{\mathcal{B}}(P_h, P_p)$ and an error boolean

$error \leftarrow 0$;
$q_p \leftarrow$ qCF for $P_p, P_h|P_s, P_t$ computed via $\mathcal{M}_k$;
$q_s \leftarrow$ qCF for $P_s, P_h|P_p, P_t$ computed via $\mathcal{M}_k$;
$q_t \leftarrow$ qCF for $P_t, P_h|P_p, P_s$ computed via $\mathcal{M}_k$;
**if** $q_p < q_s$ *and* $q_p < q_t$ **then**
    $error \leftarrow 1$;
**if** $q_s < q_t$ **then**
    **return** $P_s < P_t, error$;
**else**
    **return** $P_t < P_s, error$;

**Algorithm 4** Linearized Circular Sorting Algorithm.

**input :** Partition of species set $S$ induced by vertex $v_{\mathcal{B}}$, denoted $Part(v_{\mathcal{B}})$, Hybrid block of $Part(v_{\mathcal{B}})$, denoted $P_h$, Pivot block of $Part(v_{\mathcal{B}})$, denoted $P_p$, Set of $k$ unrooted, binary gene trees each on (a subset of) $S$

**output :** Linearized circular order of $Part(v_{\mathcal{B}})$, denoted $V_{\mathcal{B}}$, and its error score

**if** *fast mode* **then**
    Select one taxon arbitrarily from $P_i$ to use for all qCF calculations, repeating for all $i \in \{1, 2, \ldots, m_{\mathcal{B}}\}$;
    $\mathcal{M}_k \leftarrow$ algorithm for computing qCF on fly based on binary lifting algorithm;
**else**
    $\mathcal{M}_k \leftarrow$ algorithm for computing average qCF from Fig. S14 in [32]
$V_{\mathcal{B}}, error \leftarrow$ Run merge sort on $Part(v_{\mathcal{B}}) \setminus \{P_h, P_p\}$ using Algorithm 3 to compare block pairs, summing the error booleans across all comparisons;
$V_{\mathcal{B}} \leftarrow$ Append $[P_h, P_p]$ to the front of $V_{\mathcal{B}}$;
**return** $V_{\mathcal{B}}, error$

# B Software Commands

## B.1 NetCS Command

NetCS (available at `https://github.com/molloy-lab/TREE-QMC`; commit: e577f8c ) was run with command:

```
tree-qmc --network -i [input gene tree] --at [input TOB] -o [ouput]
```

## B.2 NANUQ+ Commands

NANUQ+, implemented in MSCQuartets (v3.2), was run with commands:

```
library(ape)
library(MSCquartets)
gttree <- read.tree("[input gene trees]")
QT <- quartetTableParallel(gttree, numCores = 8)
RQT <- quartetTableToRQT(QT)
pT <- quartetTreeTestInd(RQT, model=t3_model)
pT <- quartetStarTestInd(pT)
ToB <- read.tree("[input TOB]")
ToB <- labelIntNodes(ToB, plot=FALSE)
nets <- resolveLevel1(ToB, pT, alpha=[alpha], beta=[beta], plot=FALSE)
final_nwk <- nets[[1]][[1]]
```

For experiments 1–2, NANUQ+ was given the true TOB and significance thresholds: $\alpha = 10^{-7}, \beta = 0.99$ based on hyperparameter tuning the study simulating the data sets [10]. For experiment 3, NANUQ+ was given a TOB estimated with TOB-QMC and significance thresholds: $\alpha = 10^{-4}, \beta = 0.99$, as described in Section B.4.

## B.3 ASTRAL-IV Command

ASTRAL-IV (v1.23.4.6) was run with command:

```
astral4 -t 32 -o [output] [input gene trees]
```

The resulting tree was rooted at the outgroup using the scripts provided by the CAMUS study [38] with the following command:

```
python3 root-outgroup.py [ASTRAL tree]
```

## B.4 TOB-QMC commands

TOB-QMC (v4.1.5) was run with the command:

```
tree-qmc --blobsearchonly [input ASTRAL tree] \
         --store_pvalue \
         -i [input gene trees] \
         -o [TOB refinement tree]
```

to annotate the refinement tree computed with ASTRAL-IV with $p$-values using NANUQ+'s hypothesis test (T3 option). The 3f1a version of TOB-QMC was also run by adding the `-3f1a`. TOBs were constructed by contracting branches based on multiple significance thresholds with the following command:

```
tree-qmc --load_pvalue \
         --alpha [alpha] --beta 0.99 \
         -i [annotated  tree] -o [output TOB]
```

The results of this tuning experiment are shown in Fig. 5. The estimated TOBs given as input to NetCS and NANUQ+ used the 3f1a option and significance threshold $\alpha = 10^{-4}$.

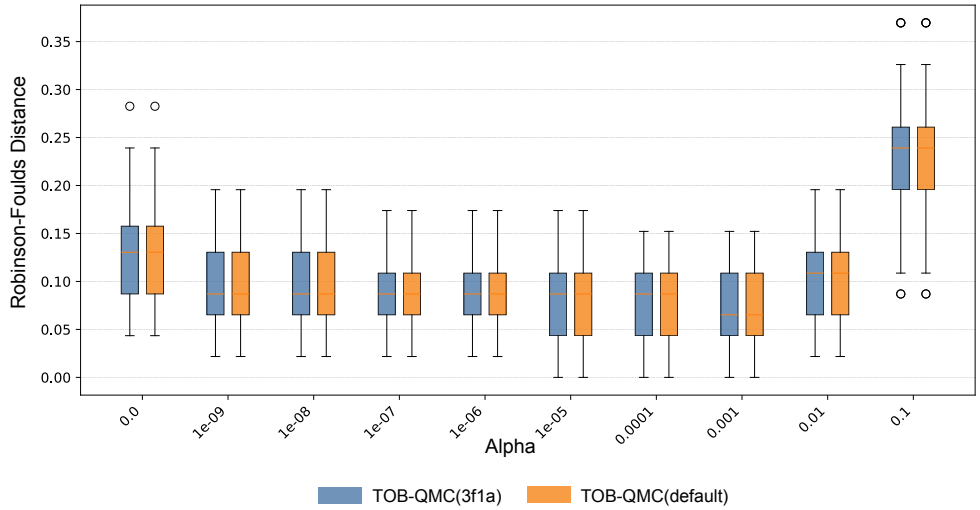



**Figure 5 Impact of the significance threshold $\alpha$ on TOB accuracy.** The $x$-axis is the significance threshold, and the $y$-axis is the normalized Robinson-Foulds distance between the true and estimated TOB. The boxes indicate which version of TOB-QMC was run, either 3f1a or default. Each boxplot has 50 replicate datasets for the 26-taxon model condition from the CAMUS study; these data sets are not used for method evaluation and benchmarking. Orange lines are medians.

## B.5 CAMUS Command

CAMUS (v1.0.0-6-g3138bcf) was run with command:

```
camus -o [output prefix] [rooted base tree] [input gene trees]
```

where the rooted base tree was computed with ASTRAL-IV.

# C Supplemental Results

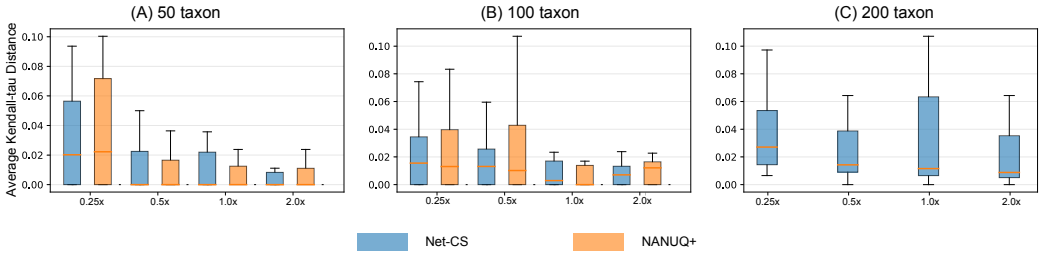



**Figure 6 Experiment 2: Average Kendall-tau distance.** Three subfigures (A), (B),(C) correspond to data sets with 50 taxa, 100 taxa, and 200 taxa, respectively. The $y$-axis is the average Kendall-tau distance, which evaluates the error in circular order reconstruction. The $x$-axis is the scaling factor: 0.25×, 0.5×, 1.0×, and 2.0× in order of decreasing ILS. Each colored box represents a method, either NetCS or NANUQ+. Both methods were given all estimated gene trees, up to 1000. For 200 taxon data, only Net-CS results are presented since NANUQ$^+$ did not complete within the maximum wallclock time of 25 hours. Orange lines are medians. Outliers are not shown.

**Table 2 Experiment 1.** Each row shows results for particular model condition defined by the number of taxa and branch scaling factor, which in turn varies ILS level. Evaluation metrics are averaged over all replicates for the same model condition. Columns indicate number of gene trees, varying from 100, 500, to 1000. Hybrid error is the fraction of incorrect hybrid across all blobs in a network. Kendall-tau distance is averaged across all blobs in a network.

| # of taxa | scale factor | Hybrid error | | | Kendall-tau distance | | | Peak RSS (MB) | | | Runtime (secs) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 100 | 500 | 1000 | 100 | 500 | 1000 | 100 | 500 | 1000 | 100 | 500 | 1000 |
| 50 | 0.25× | 0.719 | 0.412 | 0.123 | 0.149 | 0.085 | 0.034 | 273 | 300 | 331 | 4 | 5 | 5 |
| 50 | 0.5× | 0.298 | 0.096 | 0.105 | 0.049 | 0.016 | 0.012 | 273 | 299 | 330 | 7 | 4 | 6 |
| 50 | 1.0× | 0.142 | 0.100 | 0.108 | 0.017 | 0.003 | 0.020 | 274 | 298 | 329 | 4 | 3 | 4 |
| 50 | 2.0× | 0.070 | 0.053 | 0.000 | 0.020 | 0.003 | 0.003 | 274 | 297 | 328 | 4 | 3 | 4 |
| 100 | 0.25× | 0.535 | 0.114 | 0.096 | 0.140 | 0.055 | 0.032 | 282 | 332 | 395 | 13 | 9 | 16 |
| 100 | 0.5× | 0.100 | 0.025 | 0.000 | 0.070 | 0.031 | 0.016 | 283 | 332 | 395 | 8 | 11 | 13 |
| 100 | 1.0× | 0.042 | 0.025 | 0.025 | 0.028 | 0.020 | 0.011 | 283 | 332 | 395 | 7 | 7 | 10 |
| 100 | 2.0× | 0.017 | 0.000 | 0.000 | 0.008 | 0.004 | 0.005 | 285 | 334 | 395 | 5 | 8 | 9 |
| 200 | 0.25× | 0.650 | 0.042 | 0.054 | 0.168 | 0.057 | 0.034 | 300 | 398 | 524 | 41 | 40 | 65 |
| 200 | 0.5× | 0.096 | 0.037 | 0.025 | 0.054 | 0.038 | 0.025 | 299 | 399 | 525 | 25 | 31 | 55 |
| 200 | 1.0× | 0.062 | 0.025 | 0.050 | 0.045 | 0.019 | 0.019 | 302 | 401 | 526 | 27 | 25 | 49 |
| 200 | 2.0× | 0.013 | 0.025 | 0.025 | 0.015 | 0.010 | 0.011 | 303 | 403 | 528 | 19 | 22 | 46 |

# Improved Approximation Algorithms and Hardness Results for Shortest Common Superstring with Reverse Complements

**Ryosuke Yamano**
Department of Computer Science, Graduate School of Information Science and Technology, The University of Tokyo, Japan
Division of Medical Data Informatics, Human Genome Center, Institute of Medical Science, The University of Tokyo, Japan

**Tetsuo Shibuya**
Division of Medical Data Informatics, Human Genome Center, Institute of Medical Science, The University of Tokyo, Japan

## Abstract

The Shortest Common Superstring (SCS) problem is a fundamental task in sequence analysis. In genome assembly, however, the double-stranded nature of DNA implies that each fragment may occur either in its original orientation or as its reverse complement. This motivates the Shortest Common Superstring with Reverse Complements (SCS-RC) problem, which asks for a shortest string that contains, for each input string, either the string itself or its reverse complement as a substring. The previously best-known approximation ratio for SCS-RC was $\frac{23}{8}$. In this paper, we present a new approximation algorithm achieving an improved ratio of $\frac{8}{3}$. Our approach computes an optimal constrained cycle cover by reducing the problem, via a novel gadget construction, to a maximum-weight perfect matching in a general graph. We also investigate the computational hardness of SCS-RC. While the decision version is known to be NP-complete, no explicit inapproximability results were previously established. We show that the hardness of SCS carries over to SCS-RC through a polynomial-time reduction, implying that it is NP-hard to approximate SCS-RC within a factor better than $\frac{333}{332}$. Notably, this hardness result holds even for the DNA alphabet.

**2012 ACM Subject Classification** Theory of computation → Approximation algorithms analysis

**Keywords and phrases** Shortest Common Superstring, Approximation Algorithms, DNA Assembly

**Digital Object Identifier** 10.4230/LIPIcs.WABI.2026.2

**Related Version** *Previous Version*: `https://arxiv.org/abs/2603.26176`

**Funding** This work was supported by MEXT KAKENHI Grant Number 23K28035.

## 1 Introduction

The Shortest Common Superstring (SCS) problem is to find a shortest string that contains every string in a given set as a substring. It is a classical problem in combinatorial optimization and has numerous applications across a wide range of fields [6]. A particularly important application arises in DNA assembly. A DNA molecule consists of four nucleotides, namely adenine, thymine, guanine, and cytosine. The sequencing task aims to reconstruct the original molecule from a collection of short reads, which can be naturally viewed as an instance of the SCS problem over a quaternary alphabet.

However, the standard SCS formulation does not capture a crucial aspect of genome assembly, namely the double-stranded nature of DNA. Each read may originate from either strand of the DNA molecule and may therefore appear in its original orientation or as its reverse complement. Since sequencing technologies typically do not reveal the strand of origin, the target genome must contain, for each read, either the read itself or its reverse


© Ryosuke Yamano and Tetsuo Shibuya;
licensed under Creative Commons License CC-BY 4.0



complement as a substring [8]. This observation motivates the *Shortest Common Superstring with Reverse Complements* (SCS-RC) problem, which seeks a shortest string that contains, for each input string, either the string itself or its reverse complement.

Over the past decades, the standard SCS problem has been extensively studied, resulting in a rich body of literature, including a wide range of approximation algorithms [2, 1, 3, 14, 9, 12, 4, 5] as well as several inapproximability results [15, 10]. The currently best-known approximation ratio guarantee is $\frac{\sqrt{67}+14}{9} \approx 2.465$, due to Englert et al. [5], whereas the strongest inapproximability result establishes that it is NP-hard to approximate SCS within a factor better than $\frac{333}{332}$ [10].

In contrast, despite its clear relevance to genome assembly, the SCS-RC problem has received considerably less attention. Jiang et al. [7] extended the work of Blum et al. [2] and presented a 3-approximation algorithm for SCS-RC. Yamano and Shibuya [16] subsequently improved the approximation ratio to $\frac{23}{8} = 2.875$, which remains the best known. On the hardness side, Kececioglu [11] proved the NP-completeness of the decision version of SCS-RC, which is, to the best of our knowledge, the only known hardness result. These results highlight a substantial gap between the current understanding of SCS and SCS-RC. In this work, we address this gap by improving both the approximation ratio guarantees and the hardness results for SCS-RC.

## 1.1 Our Contributions

We first establish Theorem 1, improving the approximation ratio for SCS-RC from $\frac{23}{8} = 2.875$ to $\frac{8}{3} \approx 2.67$. Our result extends the classical $\frac{8}{3}$-approximation algorithm for SCS due to Breslauer et al. [3], which is based on computing a minimum-weight cycle cover, that is, a collection of vertex-disjoint directed cycles covering all vertices of a weighted directed graph encoding string overlaps in the SCS setting. In our setting, this notion is modified as follows: instead of requiring the cycle cover to include all vertices, we require that exactly one vertex is selected from each complementary pair.

Unlike in the classical SCS setting, the presence of reverse complements introduces an additional selection constraint. This constraint prevents a direct formulation as a standard cycle cover problem and precludes a straightforward reduction to weighted bipartite matching.

More generally, consider a variant of the cycle cover problem in which the vertex set is partitioned into several arbitrarily given disjoint subsets, and one is required to select exactly one vertex from each subset. The minimum-weight cycle cover problem under this constraint is known to be NP-hard [13]. Consequently, our constrained cycle cover problem cannot be reduced to this more general variant in a straightforward manner.

Although closely related cycle cover formulations have been studied in the context of reverse complements, they typically permit self-loops, which enables simple greedy algorithms [7, 16]. In contrast, excluding self-loops is essential in our setting and makes these approaches inapplicable.

We overcome these difficulties by introducing a novel gadget construction that reduces the resulting constrained minimum-weight cycle cover problem to a weighted perfect matching problem in a general graph.

▶ **Theorem 1.** *The SCS-RC problem admits an $\frac{8}{3}$-approximation algorithm.*

On the hardness side, we show that approximation hardness for SCS transfers to SCS-RC.

▶ **Theorem 2.** *If SCS-RC can be approximated within a factor of $\beta$, then SCS can also be approximated within a factor of $\beta$.*

We further show that this hardness result persists even when the alphabet size is restricted to four, corresponding to the DNA alphabet $\Sigma_{\mathrm{DNA}} = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$, where ($\mathtt{A}$, $\mathtt{T}$) and ($\mathtt{C}$, $\mathtt{G}$) are complementary pairs.

▶ **Theorem 3.** *If SCS-RC can be approximated within a factor of $\beta$ on instances over the DNA alphabet, then SCS-RC can also be approximated within a factor of $\beta$ on instances over an arbitrary alphabet.*

By combining Theorems 2 and 3 with the known hardness result that it is NP-hard to approximate SCS within a factor better than $\frac{333}{332}$ [10], we obtain Corollary 4.

▶ **Corollary 4.** *It is NP-hard to approximate SCS-RC within a factor better than $\frac{333}{332}$. This holds even for the DNA alphabet.*

## 2 Preliminaries

Throughout this paper, unless stated otherwise, we consider strings over a finite alphabet $\Sigma$ equipped with an involutive complement mapping $cm\colon \Sigma \to \Sigma$, that is, $cm(cm(x)) = x$ for every $x \in \Sigma$. A prominent example is the DNA alphabet $\Sigma_{\mathrm{DNA}} = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$, where the complement mapping is given by $cm(\mathtt{A}) = \mathtt{T}$ and $cm(\mathtt{C}) = \mathtt{G}$, reflecting the complementarity of DNA nucleotides.

For a string $s = s_1 s_2 \cdots s_n \in \Sigma^n$, its *reverse complement*, denoted by $\mathrm{rc}(s)$, is defined as

$$\mathrm{rc}(s) = cm(s_n)cm(s_{n-1}) \cdots cm(s_1).$$

This general setting allows us to capture not only the standard reverse-complement operation on DNA, but also the special case where $cm(x) = x$ for all $x \in \Sigma$, in which case $\mathrm{rc}(s)$ coincides with the usual reversal of $s$.

For a set of strings $T$, we define $\mathrm{rc}(T) = \{\mathrm{rc}(t) \mid t \in T\}$. Without loss of generality, we assume that the input set $S$ satisfies that $S \cup \mathrm{rc}(S)$ is *substring-free*, that is, no string in $S \cup \mathrm{rc}(S)$ is a substring of another string in the same set. For a string $x$, we write $x' \in \{x, \mathrm{rc}(x)\}$ to represent $x$ or its reverse complement $\mathrm{rc}(x)$. The choice of $x'$ depends on the particular solution under consideration. Let us formally define the *Shortest Common Superstring with Reverse Complements (SCS-RC)* problem.

▶ **Definition 5** (Shortest Common Superstring with Reverse Complements (SCS-RC))**.**
**Input:** *A set of strings $S = \{s_1, \ldots, s_m\}$ over $\Sigma$ such that $S \cup \mathrm{rc}(S)$ is substring-free.*
**Output:** *A shortest string $s$ such that for every $s_i \in S$, either $s_i$ or its reverse complement $\mathrm{rc}(s_i)$ appears as a substring of $s$.*

### 2.1 The Distance Graph and Cycle Cover

We follow the standard notions of *overlap*, *prefix*, and *distance* as introduced in [2]. For two strings $s$ and $t$, let $y$ be the longest string such that $s = xy$ and $t = yz$ for some non-empty strings $x$ and $z$. The length of $y$ is called the *overlap* of $s$ and $t$ and is denoted by $\mathrm{ov}(s, t)$. The string $x$ is called the *prefix* of $s$ with respect to $t$ and is denoted by $\mathrm{pref}(s, t)$. The length of $x$ is called the *distance* from $s$ to $t$ and is denoted by $\mathrm{dist}(s, t)$.

For an ordered sequence of strings $x_1, \ldots, x_r$, we define

$$\langle x_1, \ldots, x_r \rangle = \mathrm{pref}(x_1, x_2)\mathrm{pref}(x_2, x_3) \cdots \mathrm{pref}(x_{r-1}, x_r)x_r.$$

Since $S \cup \mathrm{rc}(S)$ is substring-free, there exists an optimal solution to the SCS-RC problem of the form $s = \langle s'_{q_1}, \ldots, s'_{q_m} \rangle$, where $(q_1, \ldots, q_m)$ is a permutation of $\{1, \ldots, m\}$, as observed in [2] and extended to the reverse-complement setting in [7]. Let $\mathrm{OPT}(S)$ denote the length of an optimal solution.

We consider the *distance graph* induced by the input set of strings $S$ and denote it by $G_{\mathrm{dist}}(S)$. Formally, $G_{\mathrm{dist}}(S) = (V, E, w)$ is a directed weighted graph defined as follows. The vertex set is $V = S \cup \mathrm{rc}(S)$, and the edge set is

$$E = (V \times V) \setminus \bigcup_{x \in V} \{(x, x), (x, \mathrm{rc}(x))\}.$$

Each edge $(x, y) \in E$ is assigned weight $w(x, y) = \mathrm{dist}(x, y)$.

Thus, $G_{\mathrm{dist}}(S)$ contains neither self-loops nor edges between a string and its reverse complement. This definition differs from existing distance graph constructions that take reverse complements into account [7, 16], in which self-loops are allowed. By replacing the distance function with the overlap function, we obtain the *overlap graph* $G_{\mathrm{ov}}(S)$.

For a cycle $c$ in a distance graph, let $w(c)$ denote the weight of the cycle, defined as the sum of the weights of its edges.

Given an instance $S$, consider the distance graph $G_{\mathrm{dist}}(S)$. Let $\mathrm{GTSP}(G_{\mathrm{dist}}(S))$ denote the minimum weight of a directed cycle that contains exactly one vertex from each pair $\{s_i, \mathrm{rc}(s_i)\}$ for $1 \le i \le m$. This can be viewed as a special case of the generalized traveling salesman problem (GTSP) [13], where the clusters are $\{\{s_i, \mathrm{rc}(s_i)\} \mid 1 \le i \le m\}$. Then,

$$\mathrm{GTSP}(G_{\mathrm{dist}}(S)) \le \mathrm{OPT}(S),$$

and thus provides a lower bound for the SCS-RC problem. Since GTSP is NP-hard [13], we consider a relaxation of this formulation, namely the *cycle cover* problem.

A *cycle cover* of $G_{\mathrm{dist}}(S)$ is a collection of vertex-disjoint directed cycles that contains exactly one vertex from each pair $\{s_i, \mathrm{rc}(s_i)\}$ for $1 \le i \le m$. This is a constrained variant of the classical cycle cover problem. We denote by $\mathrm{CYC}(G_{\mathrm{dist}}(S))$ a minimum-weight cycle cover, and by $w(\mathrm{CYC}(G_{\mathrm{dist}}(S)))$ its total weight. Clearly,

$$w(\mathrm{CYC}(G_{\mathrm{dist}}(S))) \le \mathrm{GTSP}(G_{\mathrm{dist}}(S)) \le \mathrm{OPT}(S).$$

A similar notion applies to the overlap graph $G_{\mathrm{ov}}(S)$. In this case, one considers a maximum-weight cycle cover, which corresponds exactly to the minimum-weight cycle cover in the distance graph $G_{\mathrm{dist}}(S)$. This correspondence follows from the relation $\mathrm{ov}(x, y) = |x| - \mathrm{dist}(x, y)$, which holds for any edge from $x$ to $y$ in the graph.

## 2.2 Periodicity of Strings

We follow the definitions in [3]. A string $x$ is a *factor* of a finite string $s$ if $s = x^i y$ for some integer $i \ge 1$ and some (possibly empty) prefix $y$ of $x$. Let $\mathrm{factor}(s)$ be the shortest such $x$, and define $\mathrm{period}(s) = |\mathrm{factor}(s)|$. A *periodic semi-infinite* string $s$ is an infinite string satisfying $s = xs$ for some nonempty string $x$. The shortest such $x$ is denoted by $\mathrm{factor}(s)$, and $\mathrm{period}(s) = |\mathrm{factor}(s)|$. Let $s$ and $t$ be strings, each of which is either finite or periodic semi-infinite. We say that $s$ and $t$ are *equivalent* if $\mathrm{factor}(s)$ and $\mathrm{factor}(t)$ are cyclic shifts of each other; that is, there exist strings $a$ and $b$ such that $\mathrm{factor}(s) = ab$ and $\mathrm{factor}(t) = ba$. Otherwise, $s$ and $t$ are said to be *inequivalent*. This defines an equivalence relation on such strings. In particular, equivalent strings have the same period.

## 3 Useful Lemmas from Previous Work

Lemma 6 relates the periodicity of overlapped strings to cycles in the distance graph. This result was proved in [2] and restated in [3].

▶ **Lemma 6** ([2]). *Let* $c = (s'_{i_1}, \ldots, s'_{i_r}, s'_{i_1})$ *be a cycle in* $\mathrm{CYC}(G_{\mathrm{dist}}(S))$. *Then*

$$w(c) = \mathrm{dist}(s'_{i_1}, s'_{i_2}) + \cdots + \mathrm{dist}(s'_{i_{r-1}}, s'_{i_r}) + \mathrm{dist}(s'_{i_r}, s'_{i_1}) = \mathrm{period}(\langle s'_{i_1}, \ldots, s'_{i_r} \rangle).$$

*Moreover, the strings* $\langle s'_{i_1}, \ldots, s'_{i_r} \rangle, \langle s'_{i_2}, \ldots, s'_{i_r}, s'_{i_1} \rangle, \ldots, \langle s'_{i_r}, s'_{i_1}, \ldots, s'_{i_{r-1}} \rangle$ *are all equivalent.*

The inequivalence of strings extracted from distinct cycles of a minimum-weight cycle cover was originally shown in [2], and later extended to the reverse-complement setting in [7, 16].

▶ **Lemma 7** ([2, 7]). *Let* $c = (s'_{i_1}, \ldots, s'_{i_r}, s'_{i_1})$ *and* $d = (s'_{j_1}, \ldots, s'_{j_k}, s'_{j_1})$ *be two distinct cycles in* $\mathrm{CYC}(G_{\mathrm{dist}}(S))$. *Define* $e = \langle s'_{i_1}, \ldots, s'_{i_r} \rangle$ *and* $f = \langle s'_{j_1}, \ldots, s'_{j_k} \rangle$. *Then the four pairs* $(e, f)$, $(e, \mathrm{rc}(f))$, $(\mathrm{rc}(e), f)$, $(\mathrm{rc}(e), \mathrm{rc}(f))$ *are all inequivalent.*

Given a semi-infinite string $\alpha = a_1 a_2 \cdots$, we denote by $\alpha[k] = a_k a_{k+1} \cdots$ its rotation starting at position $k$. Breslauer et al. [3] proved the following overlap rotation lemma.

▶ **Lemma 8** (Overlap Rotation Lemma [3]). *Let* $\alpha$ *be a periodic semi-infinite string. Then there exists an integer* $k$ *such that, for any finite string* $s$ *that is inequivalent to* $\alpha$,

$$\textit{if } \mathrm{period}(s) \le \mathrm{period}(\alpha), \textit{ then } \mathrm{ov}(s, \alpha[k]) \le \frac{2}{3}(\mathrm{period}(s) + \mathrm{period}(\alpha)).$$

We refer to a rotation $\alpha[k]$ obtained from Lemma 8 as the *critical rotation*. Lemma 5.1 of [3] was later restated in the reverse-complement setting as Lemma 14 in [16]. The inequalities in property (4) follow from the fact that, for distinct cycles $c$ and $d$, the corresponding strings $t_c$ and $t_d$ are inequivalent from property (3) and Lemma 7, while $\mathrm{period}(t_c) = w(c)$ and $\mathrm{period}(t_d) = w(d)$ from property (3) and Lemma 6.

▶ **Lemma 9** ([3, 16]). *Let* $c = (s'_{i_1}, \ldots, s'_{i_r}, s'_{i_1})$ *be a cycle in* $\mathrm{CYC}(G_{\mathrm{dist}}(S))$. *Then there exist a string* $t_c$ *and an index* $j$ *such that the following properties hold:*

**(1)** *The string* $\langle s'_{i_{j+1}}, \ldots, s'_{i_r}, s'_{i_1}, \ldots, s'_{i_j} \rangle$ *is a suffix of* $t_c$.
**(2)** *The string* $t_c$ *is a substring of* $\langle s'_{i_j}, \ldots, s'_{i_r}, s'_{i_1}, \ldots, s'_{i_j} \rangle$.
**(3)** *The string* $t_c$ *is equivalent to* $\langle s'_{i_{j+1}}, \ldots, s'_{i_r}, s'_{i_1}, \ldots, s'_{i_j} \rangle$.
**(4)** *The semi-infinite string* $\mathrm{factor}(t_c)^\infty$ *is the critical rotation of* $\mathrm{factor}(\langle s'_{i_1}, \ldots, s'_{i_r} \rangle)^\infty$.

*Moreover, let* $t_d$ *be the string obtained from this lemma corresponding to a distinct cycle* $d \in \mathrm{CYC}(G_{\mathrm{dist}}(S))$, *where* $w(d) \le w(c)$. *Then the following inequations hold:*

$$\mathrm{ov}(t_d, t_c) \le \frac{2}{3}(w(d) + w(c)), \quad \mathrm{ov}(\mathrm{rc}(t_d), t_c) \le \frac{2}{3}(w(d) + w(c)).$$

*Furthermore, the string* $t_c$ *can be computed in time linear in* $w(c)$.

Lemma 2.6 of [9] was adapted to the reverse-complement setting in Lemma 17 of [16].

▶ **Lemma 10** ([9, 16]). *Let* $T = \{t_c \mid c \in \mathrm{CYC}(G_{\mathrm{dist}}(S))\}$, *where each string* $t_c$ *is obtained as in Lemma 9. Then*

$$\mathrm{OPT}(T) \le \mathrm{OPT}(S) + w(\mathrm{CYC}(G_{\mathrm{dist}}(S))).$$

# 4 Approximation Algorithm

We begin by presenting an overview in Algorithm 1, which follows the framework of [3].

## 4.1 Overview of the Entire $\frac{8}{3}$-Approximation Algorithm

In Step 2 of Algorithm 1, the string $t_j$ contains, for each vertex of the cycle $c_j$, the string corresponding to that vertex, by Property (1) of Lemma 9. In Step 4, the resulting string is $\langle t'_{i_p}, \ldots, t'_{i_r}, t'_{i_1}, \ldots, t'_{i_{p-1}} \rangle$ if $t'_{i_p} = t_{i_p}$, and $\langle t'_{i_{p+1}}, \ldots, t'_{i_r}, t'_{i_1}, \ldots, t'_{i_p} \rangle$ otherwise. In both cases, the resulting string contains, for each vertex of the cycle $d$, the string corresponding to that vertex. Together, these two observations ensure that Algorithm 1 outputs a valid approximate solution of SCS-RC for the instance $S$.

In Step 4, we break each cycle with particular care. Specifically, we remove an edge whose overlap is bounded as guaranteed by Property (4) of Lemma 9, thereby ensuring that the increase in length incurred by breaking the cycle is properly controlled. Observe that the length of the resulting string in Step 4 is $w(d) + \mathrm{ov}(t'_{i_{p-1}}, t'_{i_p})$ when $t'_{i_p} = t_{i_p}$, and $w(d) + \mathrm{ov}(t'_{i_p}, t'_{i_{p+1}})$ otherwise.

Note that self-loops are not permitted in this step. Accordingly, self-loops are excluded in our definition of distance graphs, and hence do not appear in the minimum-weight cycle covers. The algorithmic details of computing such cycle covers are deferred to Section 4.2.

In the remainder of this subsection, we assume that the minimum-weight cycle cover can be computed in polynomial time, and focus on the approximation analysis. Specifically, we show that Algorithm 1 achieves an approximation ratio of $\frac{8}{3}$.

▶ **Theorem 11.** *Algorithm 1 yields an $\frac{8}{3}$-approximation for the SCS-RC problem.*

**Proof.** Let $\mathrm{ALG}(S)$ denote the length of the string produced by Algorithm 1. By Step 4 of the algorithm, the length $\mathrm{ALG}(S)$ can be expressed as

$$\mathrm{ALG}(S) = w(\mathrm{CYC}(G_{\mathrm{dist}}(T))) + \sum_{d \in \mathrm{CYC}(G_{\mathrm{dist}}(T))} OV_d,$$

where $OV_d$ denotes the overlap lost when breaking the cycle $d$. Combining Lemma 10 with the inequality $w(\mathrm{CYC}(G_{\mathrm{dist}}(T))) \le \mathrm{OPT}(T)$, we obtain

$$w(\mathrm{CYC}(G_{\mathrm{dist}}(T))) \le \mathrm{OPT}(S) + w(\mathrm{CYC}(G_{\mathrm{dist}}(S))).$$

We next bound the term $OV_d$ for each cycle $d$. Consider first the case where $t'_{i_p} = t_{i_p}$ in Step 4. For convenience, we define $t'_{i_0} = t'_{i_r}$ and $c_{i_0} = c_{i_r}$. In this case, the overlap lost is $\mathrm{ov}(t'_{i_{p-1}}, t'_{i_p}) = \mathrm{ov}(t'_{i_{p-1}}, t_{i_p})$. By Lemma 9, we have

$$\mathrm{ov}(t'_{i_{p-1}}, t_{i_p}) \le \frac{2}{3}\big(w(c_{i_{p-1}}) + w(c_{i_p})\big) \le \frac{2}{3} \sum_{\{j \mid t'_j \in d\}} w(c_j).$$

Now consider the case where $t'_{i_p} = \mathrm{rc}(t_{i_p})$. We define $t'_{i_{r+1}} = t'_{i_1}$ and $c_{i_{r+1}} = c_{i_1}$. The overlap lost in this case is $\mathrm{ov}(t'_{i_p}, t'_{i_{p+1}}) = \mathrm{ov}(\mathrm{rc}(t_{i_p}), t'_{i_{p+1}})$. Again by Lemma 9, we obtain

$$\mathrm{ov}(\mathrm{rc}(t_{i_p}), t'_{i_{p+1}}) = \mathrm{ov}(\mathrm{rc}(t'_{i_{p+1}}), t_{i_p}) \le \frac{2}{3}\big(w(c_{i_{p+1}}) + w(c_{i_p})\big) \le \frac{2}{3} \sum_{\{j \mid t'_j \in d\}} w(c_j).$$

In both cases, the overlap lost for cycle $d$ is bounded by $\frac{2}{3}\sum_{\{j \mid t'_j \in d\}} w(c_j)$. Summing over all cycles yields

$$\sum_{d \in \mathrm{CYC}(G_{\mathrm{dist}}(T))} OV_d \le \frac{2}{3} \sum_{d \in \mathrm{CYC}(G_{\mathrm{dist}}(T))} \sum_{\{j \mid t'_j \in d\}} w(c_j) = \frac{2}{3} w(\mathrm{CYC}(G_{\mathrm{dist}}(S))).$$

**Algorithm 1** $\frac{8}{3}$-approximation algorithm for SCS-RC.

1. Construct the distance graph $G_{\mathrm{dist}}(S)$ from the input string set $S$, and compute a minimum-weight cycle cover $\mathrm{CYC}(G_{\mathrm{dist}}(S)) = \{c_1, \ldots, c_k\}$.
2. For each $j \in \{1, \ldots, k\}$, let $t_j$ be the string obtained from the cycle $c_j$ as described in Lemma 9, and let $T = \{t_1, \ldots, t_k\}$. Construct the distance graph $G_{\mathrm{dist}}(T)$.
3. Compute a minimum-weight cycle cover $\mathrm{CYC}(G_{\mathrm{dist}}(T))$.
4. For each cycle $d = (t'_{i_1}, \ldots, t'_{i_r}, t'_{i_1})$ in $\mathrm{CYC}(G_{\mathrm{dist}}(T))$, let $p \in \{1, \ldots, r\}$ be an index such that $t'_{i_p}$ has maximum period among the vertices of $d$. If $t'_{i_p} = t_{i_p}$, break the cycle by deleting the edge incoming to $t'_{i_p}$; otherwise, delete the edge outgoing from $t'_{i_p}$. This produces a superstring containing all strings corresponding to the vertices of $d$.
5. Concatenate the resulting strings in an arbitrary order.

Combining the above bounds, we conclude that

$$\mathrm{ALG}(S) \leq \mathrm{OPT}(S) + \frac{5}{3} w(\mathrm{CYC}(G_{\mathrm{dist}}(S))) \leq \frac{8}{3}\mathrm{OPT}(S),$$

which completes the proof. ◀

## 4.2 Computing the Constrained Optimal Cycle Cover

We consider computing a maximum-weight cycle cover in the overlap graph, rather than a minimum-weight cycle cover in the distance graph. As discussed in Section 2.1, these two formulations are equivalent. However, the overlap graph admits the symmetry

$$\mathrm{ov}(s, t) = \mathrm{ov}(\mathrm{rc}(t), \mathrm{rc}(s))$$

for any strings $s$ and $t$, which simplifies the presentation of our algorithm. In the remainder of this subsection, we fix an input set of strings $S$ and consider the problem of computing a maximum-weight cycle cover in the overlap graph $G_{\mathrm{ov}}(S)$.

### Overview of the cycle cover construction

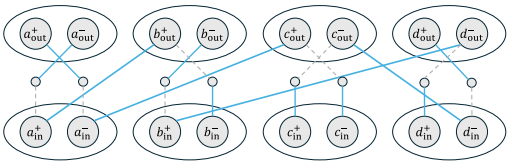
Our algorithm does not compute a cycle cover satisfying all constraints in a single step; instead, it proceeds in two stages.

In the first stage, we impose only the constraint that, for any pair of strings $s$ and $t$, the edges $s \to t$ and $\mathrm{rc}(t) \to \mathrm{rc}(s)$ cannot be selected simultaneously. This constraint is enforced via a gadget construction, under which we compute a maximum-weight perfect matching. From the resulting matching, we derive a second, symmetric matching by replacing each edge $s \to t$ with its reverse-complement counterpart $\mathrm{rc}(t) \to \mathrm{rc}(s)$. The union of these two matchings admits the extraction of a maximum-weight cycle cover satisfying the constraints, together with its reverse-complement image. The overall procedure is illustrated in Figure 1.

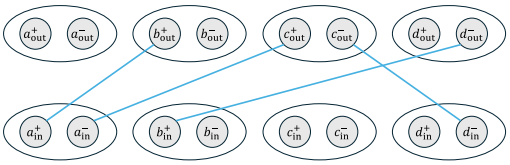
### Construction of the auxiliary graph $G'$ with gadgets

For technical convenience, we work with oriented copies of strings. For each string $s \in S$, we introduce two formal symbols $s^+$ and $s^-$, representing $s$ and its reverse complement, respectively. Let

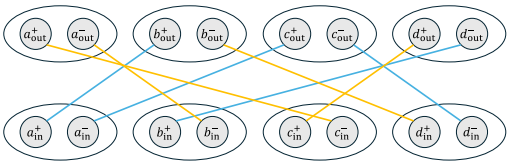
$$\tilde{S} = \bigcup_{s \in S} \{s^+, s^-\}.$$



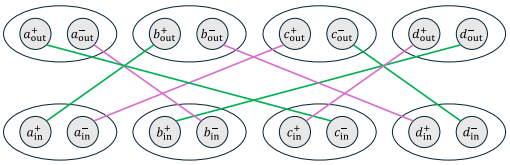
**(a)** Compute a maximum-weight perfect matching in $G'$. The small vertices in the center correspond to auxiliary vertices. The matching is shown in light blue. Dashed gray edges represent edges incident to auxiliary vertices that are not included in the matching.



**(b)** Remove all edges incident to auxiliary vertices to obtain the edge set $F$.



**(c)** Compute the reverse-complement image $\bar{F}$, shown in yellow.



**(d)** Extract two matchings $M$ and $\bar{M}$ corresponding to the resulting maximum-weight cycle cover. Matching $M$, shown in green, corresponds to the cycle $(a, \mathrm{rc}(c), \mathrm{rc}(d), b, a)$, whereas matching $\bar{M}$, shown in purple, corresponds to the cycle $(\mathrm{rc}(a), \mathrm{rc}(b), d, c, \mathrm{rc}(a))$.

**Figure 1** An example illustrating the computation of a maximum-weight cycle cover in an overlap graph induced by the string set $S = \{a, b, c, d\}$.

Even if $s = \mathrm{rc}(s)$, the two elements $s^+$ and $s^-$ are treated as distinct vertices. This convention allows us to uniformly model the problem as selecting exactly one representative from each pair $\{s^+, s^-\}$, regardless of whether $s = \mathrm{rc}(s)$ or not.

For $u = s^+$ and $v = s^-$, we define $\bar{u} = v$ and $\bar{v} = u$. More generally, for any $u \in \tilde{S}$, we denote by $\bar{u}$ the oriented copy corresponding to the reverse complement. We also associate each $u \in \tilde{S}$ with a string $\mathrm{str}(u)$, defined by

$$\mathrm{str}(s^+) = s \quad \text{and} \quad \mathrm{str}(s^-) = \mathrm{rc}(s).$$

We recall the standard reduction from cycle covers to perfect matchings. Given a directed graph, a cycle cover can be computed by reducing the problem to a perfect matching in a bipartite graph, where each vertex is split into an outgoing and an incoming copy.

As in this reduction, each vertex $u \in \tilde{S}$ is split into two vertices $u_{\mathrm{out}}$ and $u_{\mathrm{in}}$, representing outgoing and incoming edges, respectively. However, unlike the standard setting, we treat the two vertices $\{u, \bar{u}\}$ as a single cluster.

Formally, we first define a bipartite graph $\tilde{G} = (V_{\mathrm{out}}, V_{\mathrm{in}}, \tilde{E}, w)$ constructed from the overlap graph $G_{\mathrm{ov}}(S)$. The vertex sets are defined as

$$V_{\mathrm{out}} = \{u_{\mathrm{out}} \mid u \in \tilde{S}\} \quad \text{and} \quad V_{\mathrm{in}} = \{u_{\mathrm{in}} \mid u \in \tilde{S}\}.$$

The edge set $\tilde{E}$ is defined by

$$\tilde{E} = (V_{\mathrm{out}} \times V_{\mathrm{in}}) \setminus \bigcup_{u \in \tilde{S}} \{(u_{\mathrm{out}}, u_{\mathrm{in}}), (\bar{u}_{\mathrm{out}}, \bar{u}_{\mathrm{in}})\}.$$

For each edge $(u_{\mathrm{out}}, v_{\mathrm{in}}) \in \tilde{E}$, the weight is defined as $w(u_{\mathrm{out}}, v_{\mathrm{in}}) = \mathrm{ov}(\mathrm{str}(u), \mathrm{str}(v))$.

A matching in $\tilde{G}$ that corresponds to a valid cycle cover of the overlap graph $G_{\mathrm{ov}}(S)$ must satisfy the following condition. For each $u \in \tilde{S}$, exactly one of the following two cases holds:

**(i)** $u_{\mathrm{out}}$ and $u_{\mathrm{in}}$ are matched, and $\bar{u}_{\mathrm{out}}$ and $\bar{u}_{\mathrm{in}}$ are unmatched;
**(ii)** $\bar{u}_{\mathrm{out}}$ and $\bar{u}_{\mathrm{in}}$ are matched, and $u_{\mathrm{out}}$ and $u_{\mathrm{in}}$ are unmatched.

Case (i) corresponds to selecting the string $\mathrm{str}(u)$, whereas case (ii) corresponds to selecting its reverse complement $\mathrm{str}(\bar{u})$. Enforcing this constraint directly within a matching formulation, however, turns out to be technically challenging.

Instead, we impose a weaker constraint: for each $u \in \tilde{S}$, the vertices $u_{\mathrm{out}}$ and $\bar{u}_{\mathrm{in}}$ cannot be matched simultaneously. To enforce this condition, we construct a graph $G'$ by augmenting $\tilde{G}$ with the following gadget. For each $u \in \tilde{S}$, we introduce an auxiliary vertex $x_u$ and add two zero-weight edges $(u_{\mathrm{out}}, x_u)$ and $(x_u, \bar{u}_{\mathrm{in}})$.

### Obtaining the edge sets $F$ and $\bar{F}$ from a maximum-weight perfect matching in $G'$

The graph $G'$ may no longer be bipartite due to the introduction of auxiliary gadgets. Accordingly, we compute a maximum-weight perfect matching in $G'$ using an algorithm for general graphs. From the obtained matching, we discard all edges incident to auxiliary vertices and denote the remaining edge set by $F \subseteq \tilde{E}$. An example of this procedure is shown in Figures 1a and 1b. Since the matching is computed under a relaxation of the constraints required for a valid cycle cover, the total weight of $F$ is at least that of a maximum-weight cycle cover of $G_{\mathrm{ov}}(S)$.

The constraints enforced by the gadgets ensure that, for each $u \in \tilde{S}$, edges outgoing from $u_{\mathrm{out}}$ and edges incoming to $\bar{u}_{\mathrm{in}}$ are not selected simultaneously in $F$. This implies that, for any distinct $u, v \in \tilde{S}$, the two edges $(u_{\mathrm{out}}, v_{\mathrm{in}})$ and $(\bar{v}_{\mathrm{out}}, \bar{u}_{\mathrm{in}})$ cannot be selected simultaneously. Consequently, we can construct the reverse-complement image of $F$ by replacing each edge $(u_{\mathrm{out}}, v_{\mathrm{in}}) \in F$ with the edge $(\bar{v}_{\mathrm{out}}, \bar{u}_{\mathrm{in}})$. We denote the resulting edge set by $\bar{F}$. An example of this construction is shown in Figure 1c. By the symmetry of the overlap weights, namely $\mathrm{ov}(\mathrm{str}(u), \mathrm{str}(v)) = \mathrm{ov}(\mathrm{str}(\bar{v}), \mathrm{str}(\bar{u}))$, the total weight of $\bar{F}$ is equal to that of $F$. Observe that the union $F \cup \bar{F}$ is a perfect matching of $\tilde{G}$.

### Extracting two optimal cycle covers from $F \cup \bar{F}$

We prove that there exists a matching $M$ in $\tilde{G}$ corresponding to a maximum-weight cycle cover of the overlap graph $G_{\mathrm{ov}}(S)$ such that $F \cup \bar{F} = M \cup \bar{M}$, where $\bar{M}$ denotes the reverse-complement image of $M$. An example illustrating the extraction of $M$ and $\bar{M}$ is shown in Figure 1d. Since $F \cup \bar{F}$ is a perfect matching of $\tilde{G}$, it induces a collection of vertex-disjoint directed cycles covering all vertices of $\tilde{S}$. Moreover, the total weight of $F \cup \bar{F}$ is an upper bound on the total weight of $M \cup \bar{M}$, since $F$ is obtained from a relaxation of the constraints, as discussed above. Therefore, it suffices to show that each such cycle is valid with respect to the definition of a cycle cover in the original overlap graph $G_{\mathrm{ov}}(S)$.

▶ **Lemma 12.** *No cycle induced by $F \cup \bar{F}$ contains both a vertex $u \in \tilde{S}$ and its reverse-complement counterpart $\bar{u}$.*

**Proof.** Suppose, for the sake of contradiction, that there exists a directed cycle

$$c = (v_1, \ldots, \bar{v}_1, \ldots, v_1)$$

that contains both a vertex $v_1 \in \tilde{S}$ and its reverse-complement counterpart $\bar{v}_1$. Let $v_i \to v_{i+1}$ be an arbitrary directed edge of $c$.

By construction, for every edge $(e_{\mathrm{out}}, f_{\mathrm{in}}) \in F \cup \bar{F}$, its reverse-complement image $(\bar{f}_{\mathrm{out}}, \bar{e}_{\mathrm{in}})$ also belongs to $F \cup \bar{F}$. Therefore, whenever the cycle $c$ contains the edge $v_i \to v_{i+1}$ and also contains the vertex $\bar{v}_i$, it must contain the edge $\bar{v}_{i+1} \to \bar{v}_i$ as well.

Consider traversing the cycle $c$ simultaneously from $v_1$ forward and from $\bar{v}_1$ backward. Since both $v_1$ and $\bar{v}_1$ lie on the same directed cycle, the two traversals must eventually meet. At the meeting point, there exists an index $i$ such that both $v_i$ and its reverse-complement $\bar{v}_i$ appear consecutively on $c$, implying that $c$ contains a directed edge of the form $v_i \to \bar{v}_i$.

However, such an edge cannot exist: by definition of $\tilde{G}$, the edge set $\tilde{E}$ contains no edge connecting a vertex to its reverse complement. This contradiction completes the proof. ◀

By Lemma 12, for each cycle $c$ induced by $F \cup \bar{F}$, its reverse-complement image $\bar{c}$ is also induced by $F \cup \bar{F}$ and forms a distinct vertex-disjoint cycle. Hence, every induced cycle belongs to a unique pair $\{c, \bar{c}\}$, and from each pair, we can arbitrarily choose exactly one cycle. The selected cycles form a collection of vertex-disjoint directed cycles that covers exactly one vertex from each pair $\{u, \bar{u}\}$ for $u \in \tilde{S}$, and hence yields a valid cycle cover of $G_{\text{ov}}(S)$.

To conclude this section, we have shown that a maximum-weight cycle cover of the overlap graph $G_{\text{ov}}(S)$ can be computed in polynomial time. Using the equivalence between the maximum-weight cycle cover of $G_{\text{ov}}(S)$ and the minimum-weight cycle cover of $G_{\text{dist}}(S)$, this establishes Theorem 13.

▶ **Theorem 13.** *Given a distance graph $G_{\text{dist}}(S)$, the minimum-weight cycle cover $\text{CYC}(G_{\text{dist}}(S))$ can be computed in polynomial time.*

Combining Theorem 13 with Theorem 11, we obtain a polynomial-time $\frac{8}{3}$-approximation algorithm for SCS-RC. This completes the proof of our first main result, Theorem 1 from Section 1.1.

## 5 Hardness Results

In this section, we establish the hardness results stated in Theorems 2 and 3 from Section 1.1. We restate the theorems here for ease of reference.

▶ **Theorem 2.** *If SCS-RC can be approximated within a factor of $\beta$, then SCS can also be approximated within a factor of $\beta$.*

▶ **Theorem 3.** *If SCS-RC can be approximated within a factor of $\beta$ on instances over the DNA alphabet, then SCS-RC can also be approximated within a factor of $\beta$ on instances over an arbitrary alphabet.*

We first prove Theorem 2 by presenting a reduction from SCS to SCS-RC that preserves the optimal solution length. We then prove Theorem 3 by presenting a ratio-preserving reduction from SCS-RC over an arbitrary alphabet to SCS-RC over the DNA alphabet.

### 5.1 Proof of Theorem 2

Let $S = \{s_1, \ldots, s_m\}$ be an arbitrary instance of SCS, and let $A = \{a_1, \ldots, a_k\}$ denote the alphabet consisting of all characters appearing in the strings of $S$. From this instance, we construct a corresponding instance of SCS-RC as follows. Define a new alphabet

$$B = \{a_1, \ldots, a_k, a_{k+1}, \ldots, a_{2k}\},$$

and introduce a complement mapping $cm : B \to B$ defined by

$$cm(a_i) = \begin{cases} a_{i+k}, & \text{if } 1 \le i \le k, \\ a_{i-k}, & \text{if } k+1 \le i \le 2k. \end{cases}$$

It is immediate that $cm$ is an involution on $B$.

Using the alphabet $B$ equipped with the complement mapping $cm$, we consider the SCS-RC instance with the same input set $S$. We show that any approximate solution to this SCS-RC instance can be transformed into a solution that uses only strings from $S$, without increasing its length.

Let $t$ be an arbitrary approximate solution to the constructed SCS-RC instance. There exists a permutation $(i_1, \ldots, i_m)$ of $\{1, \ldots, m\}$ such that $t$ can be written as

$$t = \langle s'_{i_1}, \ldots, s'_{i_m} \rangle.$$

We define a set of intervals $I$ as follows:

$$I = \{[l, r] \mid 1 \le l \le r \le m, \text{ and } (l = 1 \text{ or } s'_{i_{l-1}} = s_{i_{l-1}}), \text{ and } (r = m \text{ or } s'_{i_{r+1}} = s_{i_{r+1}}), \\ \text{and } s'_{i_k} = \mathrm{rc}(s_{i_k}) \text{ for all } k \in \{l, \ldots, r\}\}.$$

That is, each interval $[l, r] \in I$ corresponds to a maximal substring of reverse-complemented strings in $t$. For each interval $[l, r] \in I$, the string $t$ contains the substring

$$x = \langle \mathrm{rc}(s_{i_l}), \ldots, \mathrm{rc}(s_{i_r}) \rangle.$$

We replace this substring by its reverse complement

$$\mathrm{rc}(x) = \langle s_{i_r}, \ldots, s_{i_l} \rangle.$$

Since the complement mapping $cm$ maps symbols in $A$ to symbols in $B \setminus A$ and vice versa, no overlap can occur between strings from $S$ and strings from $\mathrm{rc}(S)$. Therefore, this transformation does not decrease any existing overlap, and hence does not increase the total length of the superstring. By repeatedly applying this operation to all intervals in $I$, we eventually obtain a solution that consists solely of strings from $S$, and whose length is no greater than that of the original solution.

In particular, any optimal solution to the original SCS instance is also an optimal solution to the constructed SCS-RC instance. Moreover, any $\beta$-approximate solution to the constructed SCS-RC instance can be transformed into a solution to the original SCS instance of the same length. Hence, a $\beta$-approximation algorithm for SCS-RC yields a $\beta$-approximation algorithm for SCS.

## 5.2 Proof of Theorem 3

Let $S = \{s_1, \ldots, s_m\}$ be an arbitrary instance of SCS-RC, and let $A = \{a_1, \ldots, a_k\}$ be the alphabet consisting of all characters appearing in the strings of $S \cup \mathrm{rc}(S)$, equipped with a complement mapping $cm \colon A \to A$. Since $cm$ is an involution, for each $a \in A$, either $cm(a) = a$ or $cm(a) \neq a$ and $cm(cm(a)) = a$. Without loss of generality, we may relabel the symbols in $A$ so that there exists an index $j$ with $0 \le j \le \lfloor \frac{k}{2} \rfloor$ such that

$$cm(a_i) = \begin{cases} a_{i+j} & \text{if } 1 \le i \le j, \\ a_{i-j} & \text{if } j+1 \le i \le 2j, \\ a_i & \text{if } 2j+1 \le i \le k. \end{cases}$$

Here, we define a morphism $h \colon A^* \to \Sigma^*_{\mathrm{DNA}}$. First, for each symbol $a_i \in A$, let

$$h(a_i) = \begin{cases} \mathtt{A}^i (\mathtt{AG})^{(k+1-i-j)} \mathtt{G}^i & \text{if } 1 \le i \le j, \\ \mathtt{C}^{(i-j)} (\mathtt{CT})^{(k+1-i)} \mathtt{T}^{(i-j)} & \text{if } j+1 \le i \le 2j, \\ \mathtt{A}^{(i-j)} (\mathtt{AT})^{(k+1-i)} \mathtt{T}^{(i-j)} & \text{if } 2j+1 \le i \le k, \end{cases}$$

and extend $h$ to $A^*$ by concatenation, i.e., $h(xy) = h(x)h(y)$ for all $x, y \in A^*$. Recall that the complement mapping $cm$ on $\Sigma_{\mathrm{DNA}}$ is defined by $cm(\mathtt{A}) = \mathtt{T}$ and $cm(\mathtt{C}) = \mathtt{G}$. By construction, we have $\mathrm{rc}(h(a_x)) = h(cm(a_x))$ for every $a_x \in A$. Consequently, $\mathrm{rc}(h(s)) = h(\mathrm{rc}(s))$ holds for every string $s \in A^*$. Moreover, for any distinct symbols $a_x, a_y \in A$ with $x \neq y$, the strings $h(a_x)$ and $h(a_y)$ do not overlap. The only overlap of $h(a_x)$ with itself is the trivial one of its full length. Finally, note that $|h(a_x)| = 2(k+1-j)$ for all $a_x \in A$; hence, $|h(s)| = 2(k+1-j)\,|s|$ for every string $s \in A^*$.

Now consider the instance $R = \{h(s) \mid s \in S\}$ over $\Sigma^*_{\mathrm{DNA}}$. Let $r$ be an arbitrary approximate solution to the SCS-RC instance $R$, which can be represented by a permutation $(i_1, \ldots, i_m)$ of $\{1, \ldots, m\}$ as $r = \langle h(s'_{i_1}), \ldots, h(s'_{i_m})\rangle$. By the non-overlapping property of the encoding $h$, as shown above, that is, overlaps can occur only at the boundaries of the blocks $h(a_i)$, $r$ can be transformed into an approximate solution $s = \langle s'_{i_1}, \ldots, s'_{i_m}\rangle$ to the original instance $S$, and we have $|r| = 2(k+1-j)\,|s|$. This implies that

$$\mathrm{OPT}(R) = 2(k+1-j)\,\mathrm{OPT}(S).$$

Moreover, let $\mathrm{ALG}(R)$ denote the length of a $\beta$-approximate solution to the instance $R$. By the above argument, this solution can be transformed into a solution to the original instance $S$ of length $\mathrm{ALG}(R)/2(k+1-j)$. Therefore, we obtain

$$\frac{\mathrm{ALG}(R)/2(k+1-j)}{\mathrm{OPT}(S)} = \frac{\mathrm{ALG}(R)}{\mathrm{OPT}(R)} \leq \beta,$$

which shows that a $\beta$-approximate solution for $R$ yields a $\beta$-approximate solution for $S$.

## 6 Conclusion and Future Work

In this paper, we established an $\frac{8}{3}$-approximation algorithm for the SCS-RC problem and proved several hardness results. Our algorithm is based on computing an optimal constrained cycle cover on the distance graph, which can be viewed as a special case of the assignment-based relaxation for the generalized traveling salesman problem (GTSP), where the requirement of forming a single tour is relaxed to allow multiple disjoint cycles.

It is well known that the assignment-based relaxation of the classical traveling salesman problem reduces to finding a minimum-weight cycle cover in the standard sense, which is solvable in polynomial time. In contrast, the assignment-based relaxation of the GTSP is NP-hard, as shown in [13], where the hardness follows from a reduction to three-dimensional matching and involves clusters of size $\Theta(|V|^{\frac{2}{3}})$.

Our result demonstrates that the problem studied in this paper constitutes a nontrivial special case of the GTSP assignment-based relaxation that nevertheless admits a polynomial-time optimal algorithm. Identifying more precise structural conditions that separate tractable cases from intractable ones remains an important direction for future work.

Another important direction for future work is to further improve the approximation guarantee. One possible approach is to improve the compression ratio of the compression subroutine used in greedy-based algorithms [16]. Here, the compression ratio is defined as the ratio between the reduction from the total input length achieved by an approximate solution and that achieved by an optimal solution. For the standard SCS problem, approximation algorithms for the maximum asymmetric traveling salesman problem can be employed as black-box subroutines. In contrast, for the SCS-RC problem, one must additionally account for clusters formed by reverse-complement pairs, which constitutes a nontrivial extension of this approach. A deeper understanding of how such small clusters affect both computational complexity and approximability may lead to stronger approximation algorithms.

## References


1 Chris Armen and Clifford Stein. A 2 2/3-approximation algorithm for the shortest superstring problem. In Dan Hirschberg and Gene Myers, editors, *Combinatorial Pattern Matching*, pages 87–101, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg. doi:10.1007/3-540-61258-0_8.

2 Avrim Blum, Tao Jiang, Ming Li, John Tromp, and Mihalis Yannakakis. Linear approximation of shortest superstrings. *J. ACM*, 41(4):630–647, July 1994. doi:10.1145/179812.179818.

3 Dany Breslauer, Tao Jiang, and Zhigen Jiang. Rotations of periodic strings and short superstrings. *J. Algorithms*, 24(2):340–353, August 1997. doi:10.1006/jagm.1997.0861.

4 Matthias Englert, Nicolaos Matsakis, and Pavel Veselý. Improved approximation guarantees for shortest superstrings using cycle classification by overlap to length ratios. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2022, pages 317–330, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3519935.3520001.

5 Matthias Englert, Nicolaos Matsakis, and Pavel Veselý. Approximation Guarantees for Shortest Superstrings: Simpler and Better. In Satoru Iwata and Naonori Kakimura, editors, *34th International Symposium on Algorithms and Computation (ISAAC 2023)*, volume 283 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 29:1–29:17, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ISAAC.2023.29.

6 Theodoros P. Gevezes and Leonidas S. Pitsoulis. *The Shortest Superstring Problem*, pages 189–227. Springer New York, New York, NY, 2014. doi:10.1007/978-1-4939-0808-0_10.

7 Tao Jiang, Ming Li, and Ding-Zhu Du. A note on shortest superstrings with flipping. *Information Processing Letters*, 44(4):195–199, 1992. doi:10.1016/0020-0190(92)90084-9.

8 Eugene W. Myers Jr. A history of dna sequence assembly. *it - Information Technology*, 58(3):126–132, 2016. doi:10.1515/itit-2015-0047.

9 Haim Kaplan and Nira Shafrir. The greedy algorithm for shortest superstrings. *Information Processing Letters*, 93(1):13–17, 2005. doi:10.1016/j.ipl.2004.09.012.

10 Marek Karpinski and Richard Schmied. Improved inapproximability results for the shortest superstring and related problems. In *Proceedings of the Nineteenth Computing: The Australasian Theory Symposium - Volume 141*, CATS '13, pages 27–36, AUS, 2013. Australian Computer Society, Inc. URL: https://dl.acm.org/doi/10.5555/2525519.2525523.

11 John D. Kececioglu. *Exact and approximation algorithms for DNA sequence reconstruction*. PhD thesis, The University of Arizona, 1991. URL: https://www.proquest.com/dissertations-theses/exact-approximation-algorithms-dna-sequence/docview/304014003/se-2.

12 Marcin Mucha. Lyndon words and short superstrings. In *Proceedings of the 2013 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 958–972, 2013. doi:10.1137/1.9781611973105.69.

13 Charles E. Noon. *The Generalized Traveling Salesman Problem*. PhD thesis, The University of Michigan, 1988. URL: https://www.proquest.com/dissertations-theses/generalized-traveling-salesman-problem/docview/303673942/se-2.

14 Z. Sweedyk. A $2\frac{1}{2}$-approximation algorithm for shortest superstring. *SIAM Journal on Computing*, 29(3):954–986, 2000. doi:10.1137/S0097539796324661.

15 Virginia Vassilevska. Explicit inapproximability bounds for the shortest superstring problem. In Joanna Jędrzejowicz and Andrzej Szepietowski, editors, *Mathematical Foundations of Computer Science 2005*, pages 793–800, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. doi:10.1007/11549345_68.

16 Ryosuke Yamano and Tetsuo Shibuya. Improved Approximation Ratios for the Shortest Common Superstring Problem with Reverse Complements. In Philip Bille and Nicola Prezza, editors, *37th Annual Symposium on Combinatorial Pattern Matching (CPM 2026)*, volume 369 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 15:1–15:11, Dagstuhl, Germany, 2026. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CPM.2026.15.

# GSI: A New Approach to the Protein Inference Problem

**Aurélien Berthier**
Nantes Université, École Centrale Nantes, CNRS, LS2N, UMR 6004, F-44000 Nantes, France

**Émile Benoist**
Nantes Université, École Centrale Nantes, CNRS, LS2N, UMR 6004, F-44000 Nantes, France

**Guillaume Fertin**
Nantes Université, École Centrale Nantes, CNRS, LS2N, UMR 6004, F-44000 Nantes, France

**Géraldine Jean**
Nantes Université, École Centrale Nantes, CNRS, LS2N, UMR 6004, F-44000 Nantes, France

## Abstract

The protein inference problem, i.e., determining which proteins are present in a biological sample, is key to understanding the roles of proteins and, more broadly, many biological processes. Protein identification is typically achieved by first cleaving proteins into smaller sequences called peptides. Peptides are then identified using tandem mass spectrometry, a process that produces mass spectra, and in which peptide identification consists of associating, via dedicated tools, a mass spectrum to a peptide sequence. Protein inference consists of identifying, from a list of identified peptides, the proteins that most likely produced them, and were therefore present in the original sample.

Usually, peptide identification and protein inference are two separate steps, which are sequentially achieved. However, by proceeding in such a way, a significant amount of potentially useful information contained in the spectra may be discarded in the second step. Moreover, AI-based tools can now predict the likelihood of a peptide's identification when its parent protein is present in the sample.

In this paper, we present the Global Spectrum Interpretation (GSI) model, a protein inference model that integrates all this information to produce more accurate protein identifications. We show that GSI is NP-hard and provide a Mixed Integer Linear Program (MILP) formulation for it. This MILP is then benchmarked against state-of-the-art protein inference models on several datasets. Our results show that GSI's promising and original approach achieves performance comparable to current models and outperforms other widely used ones, while being more explainable.

**2012 ACM Subject Classification** Theory of computation → Design and analysis of algorithms; Applied computing → Computational proteomics

**Keywords and phrases** Tandem Mass Spectrometry, Peptide identification, Protein inference, Optimization, Algorithmic complexity, Mixed Integer Linear Programming

**Digital Object Identifier** 10.4230/LIPIcs.WABI.2026.3

**Supplementary Material** *Software (Source Code)*: `https://github.com/BenoistEmile/GSI`

**Funding** A. Berthier, G. Fertin, and G. Jean are supported by the French National Research Agency (ANR-24-CE45-3296), ANR PeptidOMS.

## 1 Introduction

Proteomics is the branch of biology that aims at understanding the role of proteins in living beings. An important aspect of proteomics is determining which proteins are present in a given sample. One way to do so is to use tandem mass spectrometry (MS2), in which a sample of previously digested (i.e., cleaved into smaller fragments) proteins is introduced as input, and tandem mass spectra are acquired as output. Each spectrum can be viewed as a list of peaks and represents a *peptide*, i.e., a small fragment of a protein sequence. The goal

© Aurélien Berthier, Émile Benoist, Guillaume Fertin, and Géraldine Jean;
licensed under Creative Commons License CC-BY 4.0

is to infer which proteins were present in the sample. Usually, this is done sequentially in two steps: *peptide identification*, which assigns spectra to specific peptide sequences, and *protein inference*, which relies on the identified peptides to determine which proteins were initially present in the sample [20].

### Peptide identification

The goal of peptide identification is to associate an amino acid sequence with each mass spectrum obtained during the wet-lab phase of the proteomics pipeline. Such an association is called a *peptide spectrum match* (PSM). Identifying peptides also means computing a confidence score for each PSM, which will subsequently enable statistical validation of the identifications [20].

Different approaches to peptide identification have emerged during the last few decades. *Database search* methods (e.g., X!Tandem [6], Comet [5], SpecPeptidOMS [2]) rely on proteomics databases (such as Uniprot [24]) from which candidate protein sequences are extracted. These sequences are then submitted to a step called *in silico digestion*, during which they are cleaved to generate candidate peptides. For each candidate peptide, a theoretical spectrum will then be generated and compared to the experimental spectra to identify the most likely peptide of origin for each experimental spectrum. Similar to database search methods, *spectral libraries* methods (e.g., Skyline [15]) compare experimental spectra to thoroughly vetted mass spectra collected from previous experiments. Finally, *de novo* methods (e.g., PEAKS [13]) attempt to directly reconstruct peptide sequences from experimental spectra [20].

All these methods face the issues of *scoring* (or *ranking*) and *validating* the PSMs. Although it can be done manually, the volume of data generated during MS2 experiments necessitates statistical methods capable of handling such a quantity. The PSM scores reflect the confidence of the identifications, which is usually achieved by measuring the similarity between the reference and experimental spectra. Several scoring systems exist, such as the *Shared Peaks Count* (SPC), which is the number of common peaks between both spectra, or the *hyperscore*, which refines SPC by integrating spectral peak intensity in its calculation. Probabilistic scores also exist, such as the *expectation value* (E-value), which is the probability that a given PSM is obtained by chance, or the *q-value*, which is a statistical estimate of the false discovery rate (FDR, see after).

The statistical validation is usually done using a target-decoy strategy [4], which outputs a false discovery rate (FDR) for each peptide. This strategy consists of adding so-called *decoys* to the search database. The decoys are usually generated by reversing the sequences of the *targets*, which are the peptides originally present in the search database. The FDR is computed with the following formula: $FDR(x) = \frac{N_{decoys}(x)}{N_{tot}(x)}$, where $FDR(x)$ is the FDR at a given score threshold $x$, $N_{decoys}(x)$ is the number of decoy PSMs whose scores pass the $x$ threshold, and $N_{tot}(x)$ is the total number of PSMs (targets and decoys combined) that pass the $x$ threshold. The rationale is as follows: under the hypotheses that the distribution of randomly positive identifications is the same across targets and decoys, and that all decoys identification are erroneous, the proportion of false positives among the targets at a given score threshold is the same as the proportion of identified decoys at the same threshold. The peptides are considered identified if their FDR is under a given threshold (usually 1% for peptide identification).

Peptide identification is followed by protein inference, during which the presence or absence (and eventually quantity) of candidate proteins is deduced from the set of PSMs obtained during peptide identification. As with PSMs, inferring proteins also means scoring

and validating the protein identifications. The protein inference problem is usually visualized as a bipartite graph that shows which peptides have been identified, and, for each peptide, what its parent proteins are (i.e., the proteins that can produce this peptide after being enzymatically digested). An example of such a graph is shown in the upper part of Figure 1.

### Protein inference

The goal of protein inference is then to attribute each PSM to a protein and, from these assignments, to identify which are the proteins whose presence in the sample is proven by the PSMs. An extensive review and classification of protein inference models can be found in [10]. We will focus on several categories that cover the most commonly used protein inference tools. *Bipartite graph models* only use information contained in the bipartite graph, and can be further separated into several categories. *Parsimonious* models (e.g., MaxQuant [25]) try to find the shortest list of proteins that explains the presence of all identified peptides. *Statistical* models (e.g., ProteinProphet [17], EPIFANY [18]) leverage the information contained in the graph to directly compute probabilities of presence for every candidate protein. *Optimistic* models, such as DTASelect [22], mark as present every protein that satisfies a set of conditions (such as a minimal number of identified peptides). Other approaches named supplementary information models integrate additional data compared to bipartite graph models, such as HSM [19] that uses a four-layer model directly relying on spectrum-level information to compute probabilities of presence for peptides and proteins. Finally, the recent development of Graph Neural Networks has been applied to protein inference with interesting results [14]. However, despite the increase in AI use in proteomics, few applications exist in protein inference to date.

As for PSMs, protein identifications need to be statistically validated and, to do so, scored and ranked. The protein inference tools differ in the way they score and rank protein identifications. While most methods directly estimate probabilities of presence for each protein, some do so by relying on all the PSMs associated (in the bipartite graph) with each protein, while others only rely on the one or two best-scoring PSMs. However, doing so propagates the errors made at the PSM-level to the protein identification, which makes these estimates somewhat unreliable [10]. To avoid this pitfall, *entrapment* strategies are used, in which the protein search database is expanded with so-called entrapment protein sequences, known to be absent from the sample (obtained either by shuffling the sequences from the original database, or by using proteins from organisms that are absent from the sample). The whole protein inference pipeline is then performed on the expanded database, and an FDR can be computed in a similar fashion to peptides, using entrapment proteins as decoys [26].

The number of peptides or proteins identified at a given FDR (usually 1% for peptides, and between 1% and 5% for proteins) is a widely accepted metric in the proteomics community for the comparison of peptide identification or protein inference tools.

The connection of the two successive steps (peptide identification followed by protein inference) can raise some issues. Traditionally, for each mass spectrum, only the best PSM is kept in the set transferred to the protein inference tool (and only if it gets through the validation filter). Therefore, the potentially useful information from the spectra contained in the discarded PSMs is lost. Moreover, two close peptide sequences can obtain very close scores, meaning the identification tool may not be very confident about which sequence is the correct one. However, this information would be transmitted in a very binary way to the protein inference tool, which, again, is not desirable, as shown in Figure 1. Indeed, in some cases, changing a spectrum's assignment to another peptide can lead to radically different results.

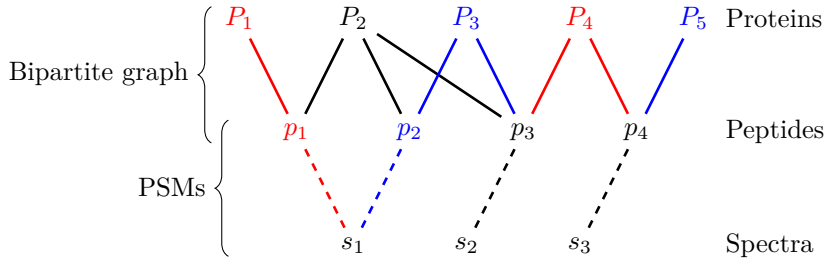



**Figure 1** Example of a (fictitious) bipartite proteins-peptides graph, with five proteins from which four peptides are digested and identified. The lower part of the figure, below the bipartite graph, depicts the spectra from which the peptides have been identified (and which peptide was identified from which spectrum). Two protein inference results obtained with a parsimonious approach (that aims at covering every identified peptide, while covering as few unidentified peptides as possible, with a minimum number of proteins) are shown, depending on the assignment of spectrum $s_1$ to peptide $p_1$ (in red) or $p_2$ (in blue). In both cases, spectrum $s_2$ (resp. $s_3$) is assigned to peptide $p_3$ (resp. $p_4$). The colored protein-peptide edges represent the assignment of each peptide to a protein in the two scenarios. Changing the assignment of a single spectrum from one peptide to another one drastically changes which proteins are inferred ($P_1$ and $P_4$ in one case, $P_3$ and $P_5$ in the other).

Integrating supplementary information while keeping the useful structure of bipartite graphs could also improve the problem's representation and help disambiguate identifications of close protein sequences. Weighting the graph's edges with *peptide detectability*, a probability (between 0 and 1) depicting how likely the identification of a given peptide is in the presence of a given protein, could be an interesting approach. Indeed, several AI-based tools are available to predict peptide detectability: DbyDeep [21] relies on bidirectional LSTMs to compute detectabilities directly from peptide sequences, and AP3 [7] uses random forest predictors to compute digestibilities (probability of tryptic cleavage) and detectabilities from 588 physicochemical features of the peptides. The accessibility of such tools makes the integration of detectability in a proteomics tool straightforward. Moreover, predicting peptide detectability and the challenges involved are gaining attention within the community, as shown by a recent review [1].

In this paper, we present a new model, called Global Spectrum Interpretation (or GSI), based on a three-layer (proteins - peptides - spectra) graph, whose goal is to simultaneously consider the two steps that are so far usually considered sequentially in proteomics (peptide identification and protein inference), while keeping useful spectral information further away in the proteomics pipeline, by eventually keeping several PSMs for one spectrum. This model can be seen as a hybrid approach between the parsimonious and optimistic models, while integrating additional information (peptide detectability, i.e., the probability that a given peptide will be identified if a given parent protein is present) to compute a more precise protein distribution. We then show that solving an instance of GSI is NP-hard and provide a Mixed Integer Linear Program (MILP) formulation to solve it. Then, we test the MILP implementation on real data from mass spectrometry experiments to evaluate its performance compared to other protein inference models. Doing so, we show that GSI outperforms other protein inference models on several metrics, thus showing encouraging results. We finally discuss the impact of the model's parameter configuration, and defend its explainability compared to other models.

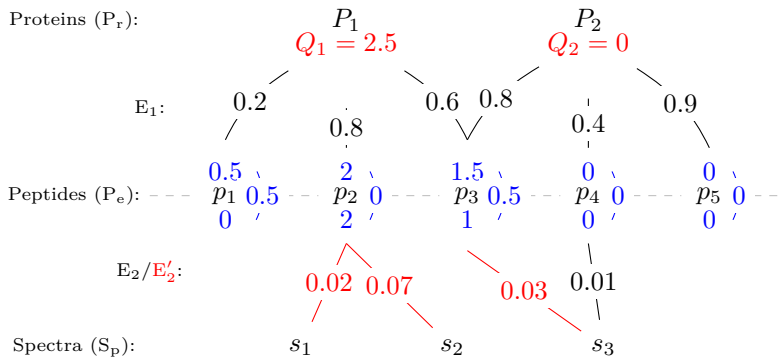



**Figure 2** An instance of the GSI problem with two proteins, five peptides, and three spectra. The elements in red present a solution for this instance, which consists of a quantity $Q_i$ per protein and an edge per spectrum (set $\mathrm{E}'_2$). The numbers in blue are the quantities $\overline{q_j}$ (over $p_j$), $\underline{q_j}$ (under $p_j$) and $\Delta_j = \overline{q_j} - \underline{q_j}$ (to the right of $p_j$). For example, $\overline{q_1} = 2.5 \times 0.2 = 0.5$, $\underline{q_1} = 0$, and $\Delta_1 = \overline{q_1} - \underline{q_1} = 0.5$.

## 2 The integrative Global Spectrum Interpretation model

The GSI model takes as input a graph $G$, whose three sets of vertices represent proteins (set $\mathrm{P_r}$), peptides (set $\mathrm{P_e}$) and spectra (set $\mathrm{S_p}$) – see Figure 2 for an illustration. $G$ contains two edge sets: the first one, $\mathrm{E}_1$, contains edges connecting each peptide to its parent proteins. These edges are weighted by the detectability of the peptides. The second edge set, $\mathrm{E}_2$, contains edges joining a spectrum to a peptide, weighted by scores that represent PSMs. In the context of proteomics, the edges in $\mathrm{E}_1$ (resp. $\mathrm{E}_2$) and the associated weights can be obtained using prediction tools (resp. peptide identification tools). Thus, an instance of our GSI model consists of a graph $G = ((\mathrm{P_r} \cup \mathrm{P_e} \cup \mathrm{S_p}), (\mathrm{E}_1 \cup \mathrm{E}_2))$, and a weight function $w$, associating a weight to each edge of the graph.

Let us now describe what a solution for GSI consists of. It can be summarized in two points: (i) choosing for each spectrum which peptide is considered to be the correct one, and (ii) assigning a quantity to the proteins (as it reflects their abundance in the original sample). For (i), we select, for each spectrum, exactly one edge incident to it, and we call this edge set $\mathrm{E}'_2$. For (ii), we assign a quantity $Q_i$ to each protein $P_i$ from $\mathrm{P_r}$. The quality of our solution relies on three elements: (a) first, the discrepancies between the distribution of peptides derived from the inferred protein quantities $Q_i$ and from the assignment of spectra using the selected edges $\mathrm{E}'_2$ should remain low; (b) second, the best PSMs should be selected, and thus the weights of the selected edges in $E'_2$ should be maximized; (c) third, the list of inferred proteins should be parsimonious, so the number of proteins with non-zero quantity should remain limited.

We present the GSI problem in the form of a minimization problem, whose objective function is a combination of these three goals. For (a), we compute, for every peptide $p_j \in \mathrm{P_e}$ the function $\Delta_j$ as follows (see also Figure 2): let us call $\underline{q_j}$ the number of edges incident to $p_j$ in $\mathrm{E}'_2$, and let $\overline{q_j}$ be the sum, over all proteins $P_i$ incident to $p_j$, of the product $Q_i \cdot w(P_i, p_j)$. We then set $\Delta_j = \overline{q_j} - \underline{q_j}$, and aim at minimizing the sum of the $\Delta_j$s under the constraint that each $\Delta_j$ must be positive. This constraint is justified by the need to explain, for each peptide identified by a spectrum, why it is present in the solution. We therefore consider that a solution unable to explain the presence of a peptide is incorrect. Concerning (b), we convert the PSM scores in such a manner that the edges in $\mathrm{E}_2$ corresponding to the best PSMs have weights closer to 0. We then aim at minimizing the global weight of the edges

selected in $E_2'$. Finally, concerning (c), we aim at minimizing the size of the set $P_r' \subseteq P_r$ of proteins with non-zero quantity (i.e., identified proteins). To reduce these three goals into a single-objective optimization problem, we combine them in the form of a weighted sum with 3 strictly positive coefficients $\Psi_1$, $\Psi_2$, and $\Psi_3$ (meant to be user-tunable). We can now formally define the GSI problem.

> Global Spectrum Interpretation (GSI)
> **Instance**: A graph $G = (P_r \cup P_e \cup S_p, E_1 \cup E_2)$, a weight function $w : (E_1 \cup E_2) \to \mathbb{R}^{*+}$, three parameters $\Psi_1, \Psi_2, \Psi_3 \in \mathbb{R}^{*+}$
> **Solution**: A quantity $Q_i \in \mathbb{R}^+$ for every protein $P_i \in P_r$, and a set of edges $E_2' \subseteq E_2$ so that each spectrum of $S_p$ is incident to exactly one edge of $E_2'$ and for every $p_j \in P_e$, $\Delta_j \geq 0$ where $\Delta_j = \overline{q_j} - \underline{q_j}$, $\overline{q_j} = \sum_{(P_i,p_j)\in E_1} w((P_i,p_j)) \cdot Q_i$ and $\underline{q_j} = \big|\{s_k | (p_j, s_k) \in E_2'\}\big|$
> **Measure**: minimize $\Psi_1 \cdot \big(\sum_{p_j \in P_e} \Delta_j\big) + \Psi_2 \cdot \big(\sum_{(p_j,s_k)\in E_2'} w((p_j,s_k))\big) + \Psi_3 \cdot |P_r'|$, where $P_r' = \{P_i \in P_r | Q_i > 0\}$

Our first result is the fact that GSI is NP-hard.

▶ **Theorem 1.** *The GSI problem is NP-hard for any set of positive parameters $\Psi_1$, $\Psi_2$ and $\Psi_3$, even in the following conditions: (i) each vertex from $P_r$ is of degree at most 3, (ii) each vertex from $P_e$ is of degree at most 4, (iii) each vertex from $S_p$ is of degree at most 3 and (iv) all edge weights are equal.*

**Proof.** The proof is by reduction from Exact Cover by 3-Sets (or X3C), which, starting from a set $\mathcal{U} = \{e_1, e_2 \dots e_m\}$ of elements (with $m = 3q$), takes as input a set $\mathcal{S} = \{S_1, S_2 \dots S_n\}$ of subsets of $\mathcal{U}$ where each $S_i$ is of cardinality 3, and asks whether it is possible to find a subset $\mathcal{S}' \subseteq \mathcal{S}$, of size $q$, such that each element $e_k$ is present exactly once in $\mathcal{S}'$. X3C is known to be NP-hard, even if every element $e_k$ appears at most 3 times in the $S_i$s [8, 11].

Starting from any instance $I_1 = (\mathcal{U}, \mathcal{S})$ of the X3C problem, we will show how to transform it into an instance $I_2 = (G = (P_r \cup P_e \cup S_p, E_1 \cup E_2), w)$ of the GSI problem. See Figure 3 for an illustration.

For each element $e_k$ of $\mathcal{U}$, we create a vertex $s_k$ in $S_p$; for each set $S_j$ of $\mathcal{S}$, we create a vertex $p_j$ in $P_e$ that we connect to all vertices $s_k$ for which $e_k \in S_j$. Next, for each vertex $p_j$ constructed in this way, we add to the graph what we call an "extension" (see the blue rectangle in Figure 3 for an example). More precisely, each vertex $p_j$ is connected by an edge to a vertex $P_j^1$ belonging to $P_r$. $P_j^1$ is itself connected to a vertex $p_j^1$ belonging to $P_e$. Three vertices $s_j^1$, $s_j^2$, and $s_j^3$ are then added to $S_p$, all being neighbors of vertex $p_j^1$. Next, three new vertices $p_j^2$, $p_j^3$, and $p_j^4$ are added to $P_e$, respectively neighbors of $s_j^1$, $s_j^2$, and $s_j^3$. To complete this extension, the vertices $p_j^2$, $p_j^3$, and $p_j^4$ are connected to a unique vertex $P_j^2$, belonging to $P_r$. All the edges in the above-described extension belong to sets $E_1$ and $E_2$ according to the vertices they connect. Finally, the weight of every edge is set to 1.

We will now show there is a solution for GSI on instance $I_2$ with score at most $\Psi_2 \cdot (|\mathcal{U}| + 3 \cdot |\mathcal{S}|) + \Psi_3 \cdot |\mathcal{S}|$, iff $I_1$ is a YES-Instance for X3C. We begin by recalling that the score of a solution to an instance of GSI is divided in three parts: (i) the sum of the $\Delta_j$s, (ii) the sum of the scores of the edges in $E_2'$ and (iii) the number of inferred proteins $|Pr'|$. Concerning score (ii) in instance $I_2$, it will be the same in any solution, as edge weights are all set to 1, leading to a sum of weights equal to $|E_2'| = |\mathcal{U}| + 3 \cdot |\mathcal{S}|$. Hence, this part of the score is $\Psi_2 \cdot (|\mathcal{U}| + 3 \cdot |\mathcal{S}|)$.

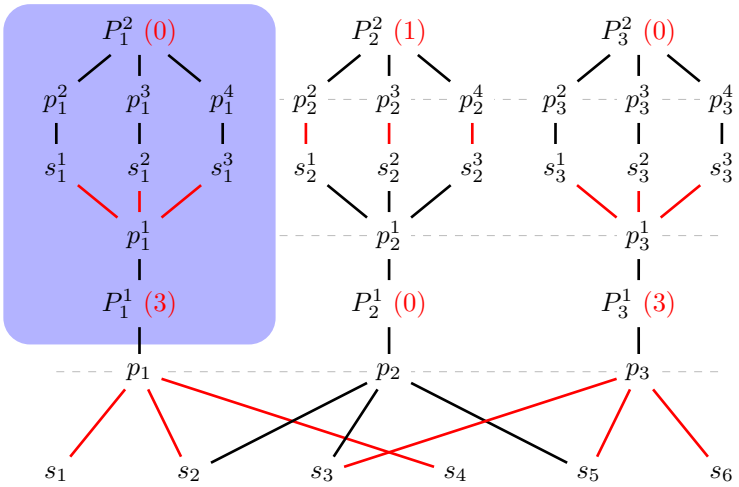



**Figure 3** Transforming the following X3C instance into an instance of GSI: $\mathcal{U} = \{e_1, e_2, e_3, e_4, e_5, e_6\}$, $\mathcal{S} = \{S_1, S_2, S_3\}$ with $S_1 = \{e_1, e_2, e_4\}$, $S_2 = \{e_2, e_3, e_5\}$, and $S_3 = \{e_3, e_5, e_6\}$. All edges have weight 1. The elements in red (edges in $E_2'$ and protein quantities) indicate an optimal solution, related to solution $\mathcal{S}' = \{S_1, S_3\}$ of X3C.

Otherwise stated, the scores from parts (i) and (iii) are the ones on which we will rely for our proof.

($\Leftarrow$) Assume that instance $I_1$ is a YES-Instance. We construct a solution for instance $I_2$ as follows: for each set $S_j = \{e_{k_1}, e_{k_2}, e_{k_3}\}$ selected in the solution $\mathcal{S}'$ for instance $I_1$, we select in $E_2'$ the edges going from $p_j$ to the three vertices $s_{k_1}$, $s_{k_2}$, and $s_{k_3}$ in $\mathrm{S_p}$. By doing this, the quantities $q_j$ are equal to 3 in the case where $S_j$ belongs to $\mathcal{S}'$, and to 0 otherwise. We then set all quantities $Q_j^1$ (concerning proteins $P_j^1$) so that the $\Delta_j$s are equal to 0: more precisely, $Q_j^1 = 3$ if $S_j \in \mathcal{S}'$, and 0 otherwise. We then select the edges incident to vertices $s_j^1$, $s_j^2$, and $s_j^3$: if $S_j \in \mathcal{S}'$, then we select in $E_2'$ the edges going from vertices $s_j^1$, $s_j^2$, and $s_j^3$ to vertex $p_j^1$; otherwise, $E_2'$ will contain the edges going from vertices $s_j^1$, $s_j^2$, and $s_j^3$ to, respectively, vertices $p_j^2$, $p_j^3$, and $p_j^4$. Having done this, we ensure that $\Delta_j^1 = 0$ for all $j$. Finally, if $S_j \in \mathcal{S}'$, then we set $Q_j^2 = 0$, otherwise we set it to 1. This last step ensures that the $\Delta_j^2$ are all equal to 0 as well. We also note that since $Q_j^1 = 3$ and $Q_j^2 = 0$ when $S_j \in \mathcal{S}'$ and $Q_j^1 = 0$, $Q_j^2 = 3$ otherwise, we have $|\mathrm{P_r}'| = |\mathcal{S}|$. The score due to the number of inferred proteins is therefore equal to $\Psi_3 \cdot |\mathcal{S}|$. Since all $\Delta_j$s are equal to 0, we have $\Psi_1 \cdot \left(\sum_{p_j \in \mathrm{Pe}} \Delta_j\right) = 0$, and the global score of the constructed solution is equal to $\Psi_2 \cdot (|\mathcal{U}| + 3 \cdot |\mathcal{S}|) + \Psi_3 \cdot |\mathcal{S}|$.

($\Rightarrow$) We now assume that there exists a solution to instance $I_2$ of the GSI problem whose score is less than or equal to $\Psi_2 \cdot (|\mathcal{U}| + 3 \cdot |\mathcal{S}|) + \Psi_3 \cdot |\mathcal{S}|$. We will show that instance $I_1$ of the X3C problem is thus a YES-Instance. First, as already discussed, note that the part of the score due to the selection of edges in $\mathrm{E_2}$ is necessarily equal to $\Psi_2 \cdot (|\mathcal{U}| + 3 \cdot |\mathcal{S}|)$. Furthermore, for each extension of the graph, either the edges going from $s_j^1, s_j^2$ and $s_j^3$ to $p_1^1$, or the edges going from the same vertices to, respectively, $p_j^2, p_j^3$ and $p_j^4$ have to be selected. The constraint $\Delta_j \geq 0$ implies that if an edge from $E_2$ incident to $p_1^1$ (resp. $p_j^2, p_j^3$, $p_j^4$) is selected, then $Q_1^1 \geq 1$ (resp. $Q_1^2 \geq 1$). Therefore, at least one protein from each extension is selected in $\mathrm{P_r}'$, so $|\mathrm{P_r}'| \geq |\mathcal{S}|$. This implies that the score of instance $I_2$ is at least $\Psi_2 \cdot (|\mathcal{U}| + 3 \cdot |\mathcal{S}|) + \Psi_3 \cdot |\mathcal{S}|$. Since we assumed that it is at most that quantity, then it must be *equal* to $\Psi_2 \cdot (|\mathcal{U}| + 3 \cdot |\mathcal{S}|) + \Psi_3 \cdot |\mathcal{S}|$, which in turn means that all $\Delta_j$s are equal to

GSI:

Objective:

$$minimize \quad \Psi_1 \cdot \left(\sum_{p_j \in \mathrm{P_e}} (\overline{q_j} - \underline{q_j})\right) + \Psi_2 \cdot \left(\sum_{(p_j,s_k)\in \mathrm{E_2}} w((p_j,s_k)) \cdot x_{(p_j,s_k)}\right) + \Psi_3 \cdot \left(\sum_{P_i \in \mathrm{P_r}} X_i\right)$$

Constraints:

| | | |
|---|---|---|
| C.1 | $\sum_{(p_j,s_k)\in \mathrm{E_2}} x_{(p_j,s_k)} = 1$ | $\forall s_k \in \mathrm{S_p}$ |
| C.2 | $Q_i \leq M \cdot X_i$ | $\forall P_i \in \mathrm{P_r}$ |
| C.3 | $\underbrace{\sum_{(P_i,p_j)\in \mathrm{E_1}} w((P_i,p_j)) \cdot Q_i - \sum_{(p_j,s_k)\in \mathrm{E_2}} x_{(p_j,s_k)}}_{\overline{q_j} - \underline{q_j} (=\Delta_j)} \geq 0$ | $\forall p_j \in \mathrm{P_e}$ |

Boundaries:

| | | |
|---|---|---|
| B.1 | $Q_i \geq 0$ | $\forall P_i \in \mathrm{P_r}$ |
| B.2 | $X_i \in \{0,1\}$ | $\forall P_i \in \mathrm{P_r}$ |
| B.3 | $x_{(p_j,s_k)} \in \{0,1\}$ | $\forall (p_j,s_k) \in \mathrm{E_2}$ |

**Figure 4** Mixed Integer Linear Program for the GSI problem.

zero. If we look at each extension of the graph (blue square in Figure 3), we see that for $\Delta_j^2$ to be zero, the edges selected in the extension in question must necessarily be either the three edges going from vertices $s_j^1$, $s_j^2$, and $s_j^3$ to vertex $p_j^1$, or the three edges going from vertices $s_j^1$, $s_j^2$, and $s_j^3$ to, respectively, vertices $p_j^2$, $p_j^3$, and $p_j^4$. In the first case, then the quantity $Q_j^2$ is necessarily 0, and in the second case, the quantity $Q_j^2$ is necessarily 1. In both cases, the quantity $q_j^1$ is either equal to 3 or equal to 0. For $\Delta_j^1$ to be equal to 0, it is therefore necessary that $\overline{Q_j^1} = 3$ (when $\underline{q_j^1} = 3$), or 0 (when $\underline{q_j^1} = 0$). Finally, if $Q_j^1$ is equal to 3 (resp. 0), then the three edges of $\mathrm{E_2}$ incident to $p_j$ are necessarily selected (resp. not selected) in $E_2'$; otherwise $\Delta_j$ would be different from 0. This implies that we can partition the vertices $s_k$ into triplets, such that, in the solution, each vertex $s_k$ of the same triplet is adjacent to the same vertex $p_j$, i.e., connected to $p_j$ by an edge of $\mathrm{E_2'}$. By construction, such a triplet contains the vertices $s_k$ corresponding to elements $e_k$ of $\mathcal{U}$, which themselves belong to the same set $S_j$. It then suffices to select, in $\mathcal{S}'$, all sets $S_j$ that contain the elements of the same triplet to obtain a solution to instance $I_1$. Therefore, $I_1$ is indeed a YES-Instance, which completes our proof. ◀

We now propose an MILP model to solve the GSI problem (Figure 4). In this model, variables $Q_i$ (see B.1) represent the quantities assigned to proteins. Boolean variables $X_i$ (see B.2) represent the fact that protein $P_i$ may be chosen ($X_i = 1$) or discarded ($X_i = 0$). In the same spirit, boolean variables $x_{(p_j,s_k)}$ (see B.3) represent the fact that edge $(p_j, s_k)$ in $\mathrm{E_2}$ may be selected ($x_{(p_j,s_k)} = 1$) or discarded ($x_{(p_j,s_k)} = 0$). Constraint C.1 forces the selection of exactly one edge for each spectrum. Constraint C.2, through the classic "big M" method, enforces quantity $Q_i$ to be equal to 0 when protein $P_i$ is not chosen (i.e., $X_i = 0$), while imposing no condition otherwise. The last constraint (C.3) corresponds to the constraint on $\Delta_j$ in the problem formulation.

## 3 Application of GSI to real data

In this section, we focus on the practical implementation of GSI. Then, we benchmark GSI against two state-of-the-art protein inference methods, X!TandemPipeline [12] (XTP) and EPIFANY [18]. Finally, we discuss the obtained results and the specificities of the model.

**Table 1** Summary of tests performed for benchmarking with GSI. From left to right, the columns indicate: the name of the dataset together with its number of spectra; the protein inference method used to compare GSI with; the peptide identification method used on the dataset together with the number of obtained PSMs; the name of the protein database as well as the total number of considered proteins; the type of evaluation method used to compare the results.

| Dataset | Inference method | Peptide identification | Protein database | Evaluation method |
|---|---|---|---|---|
| iPRG2016 (27,363 spectra) | EPIFANY | Comet + Percolator (13,457 PSMs) | iPRG2016 PrESTs + E.Coli (5,592 proteins) | Number of identifications at 3% entrapment FDR |
| HeLa (10,155 spectra) | XTP | X!Tandem (4,859 PSMs) | Human (20,422 proteins) | Number of identifications |
| HeLa (10,155 spectra) | XTP | X!Tandem (4,712 PSMs) | Swiss-Prot (568,557 proteins) | Number of human identifications |

## 3.1 GSI implementation and parameters

To allow further experimental studies about GSI, the proposed MILP was implemented in C++ using the CPLEX Optimizer (`https://www.ibm.com/fr-fr/products/ilog-cplex-optimization-studio`). Our implementation can be found on the following link: `https://github.com/BenoistEmile/GSI`.

All the following experiments have been done using a standard laptop (Intel Core Ultra 7 165H, 16 cores, 64GB RAM, Windows 11 25H2).

Detectabilities have been computed either with AP3 [7] or with DbyDeep [21], depending on the dataset under study.

Parameters $\Psi_1, \Psi_2, \Psi_3$ do not have default values. Instead, they have been set differently depending on the dataset at hand, after a series of tests to determine the ones that are favorable for GSI. This is also discussed, concerning dataset iPRG2016, in Section 3.3.

## 3.2 GSI benchmarking

In this section, we present a benchmark comparison between GSI and two well-established protein inference tools, which both rely on a statistical framework: X!TandemPipeline was selected because it is a convenient tool that integrates both peptide identification (using the widely-used X!Tandem algorithm) and protein inference, while EPIFANY is a more recent protein inference model that demonstrated strong performance on the iPRG2016 benchmark dataset [23]. Table 1 provides a comprehensive overview of the comparisons presented and detailed in the following paragraphs. While several protein inference models were presented in the Introduction, it was not possible to compare all of them. We therefore selected two reference models (X!TandemPipeline and EPIFANY) for our benchmark, but it should be noted that GraphPI achieved results close to EPIFANY on the iPRG2016 dataset that we used [14].

**Table 2** Number of proteins identified by XTP and GSI on the human and Swiss-Prot databases. The ∩ column corresponds to the number of proteins that were identified by both tools. The "Groups" row indicates the number of protein groups identified by XTP. The "Human proteins" row indicates the number of human proteins among the identified proteins on the multi-species database.

| | Human Database | | | Swiss-Prot database | | |
|---|---|---|---|---|---|---|
| | XTP | ∩ | GSI | XTP | ∩ | GSI |
| Identifications | 690 | 642 | 676 | 1965 | 366 | 868 |
| Groups (XTP) | 654 | | | 653 | | |
| Human proteins | | | | 625 (32%) | 283 (77%) | 330 (38%) |

### Comparison with X!TandemPipeline

To compare GSI and XTP, we used a dataset composed of spectra obtained from a lysate of HeLa cells. We conducted two experiments on this dataset's spectra (lines 2 and 3, Table 1). In both experiments, the well-established database search engine X!Tandem [3], which is embedded in XTP, is used to provide PSMs, scored with X!Tandem's hyperscore. What distinguishes the experiments is the protein database used to query spectra. The first database is composed of the 20,422 human proteins present in the Uniprot/Swiss-Prot database (`https://www.uniprot.org`). Since the spectra were derived from human cells, all the proteins in this database were potentially present in this sample, although it was not known exactly which ones. Hence, XTP and GSI are evaluated based on the number of proteins they are capable of identifying. The second database is composed of the 568,557 proteins present in Swiss-Prot, all species combined. Using this database allowed us to compare the proportion of identifications of non-human proteins (which are necessarily wrong identifications), and therefore the ability of the two engines to produce reliable identifications. In GSI, peptide detectabilities were computed using AP3 [7] and parameters $\Psi_1, \Psi_2$ and $\Psi_3$ were all set to 1.

GSI and XTP were then run on the obtained PSMs. GSI achieved a mean runtime of 56 seconds (over 5 runs) on the Swiss-Prot database, using a standard laptop, which is comparable to what was achieved by XTP. The detailed numbers of identifications from the two engines are provided in Table 2. On the human database, XTP identifies 654 protein groups regrouping 690 proteins in total. A group identification is produced when XTP is unable to distinguish between several proteins (for example, because they share the same subset of identified peptides). Therefore, the number of proteins whose presence has been proven by XTP is the number of group identifications. GSI identifies 676 proteins, of which 642 are identified by both models. This shows that GSI can produce reliable identifications and even outperforms XTP by identifying 3% more proteins.

On the multi-species database, XTP identifies 1,965 proteins in 653 groups, while GSI identifies 868 proteins, of which 366 are also identified by XTP. Among the 1,965 proteins identified by XTP, 625 are human proteins, yielding 32% of correct identifications. GSI identifies 330 of human proteins, yielding 38% of correct identifications. XTP identifies almost three times more individual proteins than in the first experiment, but the number of groups remains very close. On the other hand, GSI identifies more proteins than in the first experiment, but the increase is better controlled than for XTP (increase of only 28.4%). This suggests that GSI is able to better distinguish between similar proteins, which one is truly present in the sample. It can be explained by the fact that XTP uses a grouping

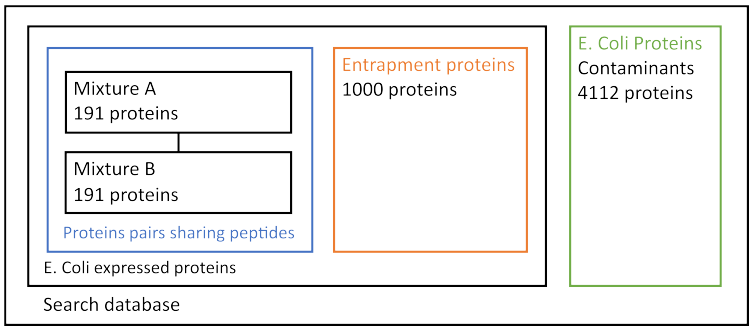



**Figure 5** The proteins contained in the iPRG2016 search database belong to one of the following categories: proteins from mixtures A and B ; entrapment proteins, which are close to mixtures A and B proteins but are absent from the sample; contaminant proteins, which are likely to be present in the sample given its preparation protocol, but are not of interest. As we study a sample only containing mixture A proteins, we compute the entrapment FDR using mixture A proteins as targets and mixture B and entrapment proteins as decoys.

strategy without additional information, and hence is unable to distinguish which protein of each protein group is truly present in the sample. By contrast, GSI does not use this approach and instead integrates peptide detectability, which may explain (at least in part) its improved results. Furthermore, GSI identifies proportionally 18.8% more human proteins than XTP, which corroborates that GSI better distinguishes correct (in this case, human proteins) from incorrect (non-human proteins) identifications.

## Comparison with EPIFANY

To benchmark GSI against EPIFANY, we chose to use the iPRG2016 dataset, which is designed for benchmarking protein inference, and in particular, to evaluate their ability to distinguish proteins whose sequences are close (see Table 1, line 1). EPIFANY was already benchmarked on this dataset, making it particularly convenient to compare the two models. We followed the same protocol as described in [18]. This protocol uses a search database composed of 4 categories of proteins (Figure 5): 191 *Mixture A* proteins, that are present in the analysed sample and that should be identified; 191 *Mixture B* proteins, that each share one peptide with mixture A proteins and are absent from the sample; 1,000 *entrapment* proteins, that are similar to mixture A and B proteins and are absent from the sample; 4,112 contaminant proteins, added to the database to avoid identifying proteins resulting from sample contamination. We downloaded the PSM identifications, scored by q-values, a statistical estimate of the FDR, as well as the protein identifications from EPIFANY. We computed peptide detectabilities for each identified peptide using DbyDeep [21] and set parameters $\Psi_1 = 1$, $\Psi_2 = 1$, and $\Psi_3 = 100$. We then applied GSI to the obtained PSMs, filtered at a 1% FDR (which represents 13,457 PSMs) as was done by EPIFANY, and obtained raw protein identifications. GSI achieved a mean runtime of 37 seconds (over 5 runs) on this dataset, using a standard laptop. For comparison, the reported runtime, in the paper, for EPIFANY on a similar number of peptides and proteins, and with a similar computer, is around 100 seconds. We used the same entrapment strategy as described in [18] (and used to validate EPIFANY's identifications) to compute an FDR and validate GSI's identifications. To compute the FDR used to validate the identifications, these proteins have to be labelled as *target* or *decoy*. The mixture A proteins, which we wish to identify, are labelled as targets. The mixture B and entrapment proteins are labelled as decoys.

**Table 3** Number of proteins identified by EPIFANY and GSI on the iPRG2016 dataset, after filtering at a 3% entrapment FDR threshold. The ∩ column corresponds to the number of proteins that were identified by both tools.

| | EPIFANY | ∩ | GSI |
|---|---|---|---|
| Identifications (at 3% FDR) | 176 | 171 | 176 |

The identifications were filtered at a 3% entrapment FDR threshold for both models. The detailed numbers of identifications from the two engines are provided in Table 3. Both models identified 176 proteins, with an overlap of 171 proteins, thus showing that the two models have very close results. The benchmarking results on the iPRG2016 dataset therefore show that GSI can identify as many proteins as EPIFANY, with a runtime reduced by roughly two-thirds. Moreover, among the 1,681 spectra that were associated with several PSMs, 568 had PSMs with different scores. The model assigned 185 of these spectra (33%) to a peptide that was not the best-scoring one.

## 3.3 Discussion

The results of the benchmarks on both datasets show that GSI identifies as many proteins as a recent protein inference model (EPIFANY), with a considerably shorter runtime, and even outperforms an established tool (XTP). As seen earlier, most of GSI's comparative advantage likely derives from its use of peptide detectability, which allows it to disambiguate the identifications of similar proteins rather than regrouping them in common group identifications. Other approaches, such as the use of multiple digestion enzymes [16], can also perform such disambiguation but at a higher sample processing cost.

The analysis of GSI's performance is, however, hindered by the datasets used for the benchmarks. Indeed, the HeLa dataset provided, for most spectra, only one PSM per spectrum. With this dataset, it is therefore impossible to understand to what extent GSI's flexibility in assigning spectra to peptides is impacting the protein identification results. The iPRG2016 dataset, used for benchmarking GSI against EPIFANY, provides several PSMs for a higher number of spectra, and the benchmark on this dataset shows that GSI is able to provide a convincing solution while relying on less confidently identified peptides. However, only 5% of the identified spectra had multiple PSMs with discriminative scores (i.e., not all equal), which limits our understanding of the impact of this aspect of the modeling.

Moreover, in GSI, the non-probabilistic model associated with the constraints we set ensures that every protein identification can be explained by specific PSMs, thus providing a sounder interpretation than probabilistic models, which, although they are currently the most used, lack explainability.

Additionally, as shown in Table 4, we performed a benchmark concerning parameters $\Psi_1$, $\Psi_2$ and $\Psi_3$, on the iPRG2016 dataset.

We first tried to understand the relative weights of the three objectives. Indeed, the orders of magnitude of the three objectives are very different: in the case of the iPRG2016 dataset, the first objective is computed from the $\Delta$ values on up to 13,457 peptides, the second one is computed from the scores that are q-values (thus all $< 0.01$) of around 500 spectra (all the other spectra have only one PSM, or PSMs with equal values, and thus have no impact on this objective value), and the third one can go up to 5,592 if all proteins are identified. We therefore tried several configurations of the $\Psi$ parameters, which show that the best results are achieved when $\Psi_1 = 1$ and $\Psi_3 = 100$. This can be expected because, on

this dataset, the main difficulty is to identify as few entrapment proteins (False Positives) as possible. Increasing the weight of the parsimony constraint encourages the model to identify fewer proteins, thus, thanks to the other constraints, focusing its identifications on correct proteins, and consequently lowering the number of false positives and improving the results. Nevertheless, the results are identical whether $\Psi_2 = 1$ or $\Psi_2 = 100$. We scrutinized the impact of the corresponding objective by deactivating it ($\Psi_2 = 0$) or setting an intermediate value ($\Psi_2 = 10$) (see Table 4, $\Psi_2$ impact). In both cases, the results are still the same. However, as explained above, this objective's value has an amplitude of 5, while the third objective will increase by 100 for each additional inferred protein. This is why it is logical that the value of $\Psi_2$ does not influence the results in that case. However, the flexibility offered by GSI on the assignment of spectra to peptides not only impacts the $\Psi_2$ objective, but also the computation of the $\Delta$ values, and thus the $\Psi_1$ objective, which has been shown to be important in this case.

Finally, we studied the impact of deactivating one or two objectives by setting the corresponding $\Psi$ parameters to 0 (see Table 4, deactivation of one or two objectives). None of the tested configurations obtained an F-Score and a number of identified proteins as high as the $\Psi_1 = 1$, $\Psi_3 = 100$ configuration. The only case where an objective could be deactivated while maximizing the F-Score and the number of identifications is $\Psi_1 = 1$, $\Psi_2 = 0$, $\Psi_3 = 100$, for the reason aforementioned.

Using a multi-objective optimization approach could also be interesting in the future, to better understand our model by circumventing the use of such parameters, and directly identifying the best configuration.

## 4 Conclusion

In this paper, we presented and discussed an optimization problem motivated by proteomics, and more precisely, protein inference in mass spectrometry, that simultaneously optimizes two previously separated steps, peptide identification and protein inference. We showed that GSI is NP-hard even in very restricted conditions, and provided an MILP formulation for it. The application of this MILP to real datasets appears to be fast, even on large ones. Moreover, our model showed promising results, as presented by their comparison to the results obtained by XTP and EPIFANY, and has the advantage of obtaining results that are better explicable than the probabilistic models we compared it with.

Nevertheless, there is still room for improvement. Our GSI model could indeed be further developed to account for additional biological features, such as chimeric spectra, that represent up to 50% of the spectra in large proteomics datasets [9]. For this type of spectrum, which is produced by two different peptides, it would be interesting to modify GSI so that it assigns one spectrum to up to two peptides. Another way to improve GSI is to focus on its objective function, which is currently a weighted sum of three separate objectives. Converting GSI into a multi-objective problem using these three objective functions could also prevent us from using parameters $\Psi_1$, $\Psi_2$, and $\Psi_3$, whose impact is not yet fully understood, as previously discussed. On this specific point, we are planning a new implementation of GSI using a multiobjective solver, such as benpy (`https://pypi.org/project/benpy/`), to study this approach. This new implementation also aims to be easier to use and integrate into pipelines than the current one.

**Table 4** Number of identifications using different configurations of the $\Psi$ parameters on the iPRG2016 dataset. The "TP", "FP", "FN", and "TN" columns respectively contain the number of true positives, false positives, false negatives, and true negatives for each configuration. The F-Score, defined by $F_1 = \frac{2TN}{2TN+FP+FN}$, is a measure of the predictions' quality. The last column indicates the number of proteins identified after filtering the results at 3% entrapment FDR. The best F-score and the highest number of identifications achieved are shown in bold.

| $\Psi_1$ | $\Psi_2$ | $\Psi_3$ | TP | FP | FN | TN | F-Score | Number of identifications at 3% FDR |
|---|---|---|---|---|---|---|---|---|
| **Relative importance of the three objectives** | | | | | | | | |
| 1 | 1 | 1 | 176 | 47 | 16 | 1077 | 0.85 | 141 |
| 100 | 1 | 1 | 176 | 56 | 16 | 1068 | 0.83 | 140 |
| 1 | 100 | 1 | 176 | 46 | 16 | 1078 | 0.85 | 141 |
| 1 | 1 | 100 | 176 | 6 | 16 | 1118 | **0.94** | **176** |
| 1 | 100 | 100 | 176 | 6 | 16 | 1118 | **0.94** | **176** |
| 100 | 1 | 100 | 176 | 47 | 16 | 1077 | 0.85 | 141 |
| 100 | 100 | 1 | 176 | 56 | 16 | 1068 | 0.83 | 140 |
| **$\Psi_2$ impact** | | | | | | | | |
| 1 | 0 | 100 | 176 | 6 | 16 | 1118 | **0.94** | **176** |
| 1 | 10 | 100 | 176 | 6 | 16 | 1118 | **0.94** | **176** |
| **Deactivation of one objective** | | | | | | | | |
| 0 | 1 | 1 | 172 | 9 | 20 | 1115 | 0.92 | 172 |
| 1 | 0 | 1 | 176 | 49 | 16 | 1075 | 0.84 | 141 |
| 1 | 1 | 0 | 176 | 56 | 16 | 1068 | 0.83 | 140 |
| **Deactivation of two objectives** | | | | | | | | |
| 1 | 0 | 0 | 176 | 56 | 16 | 1068 | 0.83 | 140 |
| 0 | 1 | 0 | 176 | 21 | 16 | 1103 | 0.90 | 149 |
| 0 | 0 | 1 | 172 | 9 | 20 | 1115 | 0.92 | 172 |

## References


1 Jesse Angelis, Eva Ayla Schröder, Zixuan Xiao, Wassim Gabriel, and Mathias Wilhelm. Peptide Property Prediction for Mass Spectrometry Using AI: An Introduction to State of the Art Models. *PROTEOMICS*, 25(9-10):e202400398, 2025. doi:10.1002/pmic.202400398.

2 Émile Benoist, Géraldine Jean, Hélène Rogniaux, Guillaume Fertin, and Dominique Tessier. SpecPeptidOMS Directly and Rapidly Aligns Mass Spectra on Whole Proteomes and Identifies Peptides That Are Not Necessarily Tryptic: Implications for Peptidomics. *Journal of Proteome Research*, March 2025. doi:10.1021/acs.jproteome.4c00870.

3 Robertson Craig and Ronald C. Beavis. TANDEM: Matching proteins with tandem mass spectra. *Bioinformatics*, 20(9):1466–1467, June 2004. doi:10.1093/bioinformatics/bth092.

4 Joshua E. Elias and Steven P. Gygi. Target-decoy search strategy for increased confidence in large-scale protein identifications by mass spectrometry. *Nature Methods*, 4(3):207–214, March 2007. doi:10.1038/nmeth1019.

5 Jimmy K. Eng, Tahmina A. Jahan, and Michael R. Hoopmann. Comet: An open-source MS/MS sequence database search tool. *PROTEOMICS*, 13(1):22–24, 2013. doi:10.1002/pmic.201200439.

6 David Fenyö and Ronald C. Beavis. A Method for Assessing the Statistical Significance of Mass Spectrometry-Based Protein Identifications Using General Scoring Schemes. *Analytical Chemistry*, 75(4):768–774, February 2003. doi:10.1021/ac0258709.

**7** Zhiqiang Gao, Cheng Chang, Jinghan Yang, Yunping Zhu, and Yan Fu. AP3: An Advanced Proteotypic Peptide Predictor for Targeted Proteomics by Incorporating Peptide Digestibility. *Analytical Chemistry*, 91(13):8705–8711, July 2019. doi:10.1021/acs.analchem.9b02520.

**8** Michael R Garey and David S Johnson. *Computers and intractability*, volume 174. Freeman San Francisco, 1979.

**9** Stephane Houel, Robert Abernathy, Kutralanathan Renganathan, Karen Meyer-Arendt, Natalie G. Ahn, and William M. Old. Quantifying the Impact of Chimera MS/MS Spectra on Peptide Identification in Large-Scale Proteomics Studies. *Journal of Proteome Research*, 9(8):4152–4160, August 2010. doi:10.1021/pr1003856.

**10** Ting Huang, Jingjing Wang, Weichuan Yu, and Zengyou He. Protein inference: A review. *Briefings in Bioinformatics*, 13(5):586–614, September 2012. doi:10.1093/bib/bbs004.

**11** R.M. Karp. Reducibility Among Combinatorial Problems. *Complexity of Computer Computations*, pages 85–103, 1972.

**12** Olivier Langella, Benoît Valot, Thierry Balliau, Mélisande Blein-Nicolas, Ludovic Bonhomme, and Michel Zivy. X!TandemPipeline: A Tool to Manage Sequence Redundancy for Protein Inference and Phosphosite Identification. *Journal of Proteome Research*, 16(2):494–503, February 2017. doi:10.1021/acs.jproteome.6b00632.

**13** Bin Ma, Kaizhong Zhang, Christopher Hendrie, Chengzhi Liang, Ming Li, Amanda Doherty-Kirby, and Gilles Lajoie. PEAKS: Powerful software for peptide de novo sequencing by tandem mass spectrometry. *Rapid Communications in Mass Spectrometry*, 17(20):2337–2342, 2003. doi:10.1002/rcm.1196.

**14** Zheng Ma, Jiazhen Chen, Lei Xin, and Ali Ghodsi. GraphPI: Efficient Protein Inference with Graph Neural Networks. *Journal of Proteome Research*, 23(11):4821–4834, November 2024. doi:10.1021/acs.jproteome.3c00845.

**15** Brendan MacLean, Daniela M. Tomazela, Nicholas Shulman, Matthew Chambers, Gregory L. Finney, Barbara Frewen, Randall Kern, David L. Tabb, Daniel C. Liebler, and Michael J. MacCoss. Skyline: An open source document editor for creating and analyzing targeted proteomics experiments. *Bioinformatics*, 26(7):966–968, April 2010. doi:10.1093/bioinformatics/btq054.

**16** Rachel M. Miller, Robert J. Millikin, Connor V. Hoffmann, Stefan K. Solntsev, Gloria M. Sheynkman, Michael R. Shortreed, and Lloyd M. Smith. Improved Protein Inference from Multiple Protease Bottom-Up Mass Spectrometry Data. *Journal of Proteome Research*, 18(9):3429–3438, September 2019. doi:10.1021/acs.jproteome.9b00330.

**17** Alexey I. Nesvizhskii, Andrew Keller, Eugene Kolker, and Ruedi Aebersold. A Statistical Model for Identifying Proteins by Tandem Mass Spectrometry. *Analytical Chemistry*, 75(17):4646–4658, September 2003. doi:10.1021/ac0341261.

**18** Julianus Pfeuffer, Timo Sachsenberg, Tjeerd M. H. Dijkstra, Oliver Serang, Knut Reinert, and Oliver Kohlbacher. EPIFANY: A Method for Efficient High-Confidence Protein Inference. *Journal of Proteome Research*, 19(3):1060–1072, March 2020. doi:10.1021/acs.jproteome.9b00566.

**19** Changyu Shen, Zhiping Wang, Ganesh Shankar, Xiang Zhang, and Lang Li. A hierarchical statistical model to assess the confidence of peptides and proteins inferred from tandem mass spectrometry. *Bioinformatics*, 24(2):202–208, January 2008. doi:10.1093/bioinformatics/btm555.

**20** Steven R. Shuken. An Introduction to Mass Spectrometry-Based Proteomics. *Journal of Proteome Research*, 22(7):2151–2171, July 2023. doi:10.1021/acs.jproteome.2c00838.

**21** Juho Son, Seungjin Na, and Eunok Paek. DbyDeep: Exploration of MS-Detectable Peptides via Deep Learning. *Analytical Chemistry*, 95(30):11193–11200, August 2023. doi:10.1021/acs.analchem.3c00460.

**22** David L. Tabb, W. Hayes McDonald, and John R. Yates. DTASelect and Contrast: Tools for Assembling and Comparing Protein Identifications from Shotgun Proteomics. *Journal of Proteome Research*, 1(1):21–26, February 2002. doi:10.1021/pr015504q.

23 Matthew The, Fredrik Edfors, Yasset Perez-Riverol, Samuel H. Payne, Michael R. Hoopmann, Magnus Palmblad, Björn Forsström, and Lukas Käll. A Protein Standard That Emulates Homology for the Characterization of Protein Inference Algorithms. *Journal of Proteome Research*, 17(5):1879–1886, May 2018. doi:10.1021/acs.jproteome.7b00899.

24 The UniProt Consortium. UniProt: The Universal Protein Knowledgebase in 2025. *Nucleic Acids Research*, 53(D1):D609–D617, January 2025. doi:10.1093/nar/gkae1010.

25 Stefka Tyanova, Tikira Temu, and Juergen Cox. The MaxQuant computational platform for mass spectrometry-based shotgun proteomics. *Nature Protocols*, 11(12):2301–2319, December 2016. doi:10.1038/nprot.2016.136.

26 Bo Wen, Jack Freestone, Michael Riffle, Michael J. MacCoss, William S. Noble, and Uri Keich. Assessment of false discovery rate control in tandem mass spectrometry analysis using entrapment. *Nature Methods*, 22(7):1454–1463, July 2025. doi:10.1038/s41592-025-02719-x.

# Contig Model for Variable-Order de Bruijn Graphs

**Diego Díaz-Domínguez**
Department of Computer Science, University of Helsinki, Finland

**Pierfrancesco Martinello**
Department of Computer Science, University of Helsinki, Finland

**Taku Onodera**
Graduate School of Information Science and Technology, University of Tokyo, Japan

**Simon J. Puglisi**
Department of Computer Science, University of Helsinki, Finland

**Leena Salmela**
Department of Computer Science, University of Helsinki, Finland

## Abstract

Choosing an order for constructing a *de Bruijn graph* (DBG) is a crucial step in *de novo* assembly, as no single value allows complete genome reconstruction. The *variable-order de Bruijn graph* (voDBG) addresses this limitation by combining DBGs of multiple orders in a single structure connected by contextual relationships. This representation enables new connections to be identified or ambiguities to be resolved during assembly. However, voDBGs currently lack a formal definition of contigs.

In this paper, we give the first formal definition of contigs for voDBGs. We show that, for a frequency range $[\ell, h]$ with $\ell > h/2$, nodes whose labels occur with frequency $f \in [\ell, h]$ in the reads spell sequences of the genome with high probability under uniform sampling assumptions. We call these sequences $(\ell, h)$-tigs. We also present an efficient algorithm to enumerate $(\ell, h)$-tigs from a voDBG that accounts for homopolymer errors. Experiments on PacBio HiFi data show that our method significantly improves contiguity compared to unitigs in fixed-order DBGs while remaining considerably lighter than full genome assemblers.

**2012 ACM Subject Classification** Applied computing → Bioinformatics; Applied computing → Molecular sequence analysis; Theory of computation → Data structures design and analysis; Applied computing → Computational genomics

**Keywords and phrases** genome assembly, de Bruijn Graph, long reads

**Digital Object Identifier** 10.4230/LIPIcs.WABI.2026.4

**Related Version** *Preprint Version*: `https://doi.org/10.1101/2022.09.06.506758`

**Supplementary Material** *Software (Source Code)*: `https://github.com/ddiazdom/ryu`

**Funding** *Diego Díaz-Domínguez*: supported by EU Horizon Europe grant No 101060011 and Research Council of Finland via grant 323233.
*Pierfrancesco Martinello*: supported by Research Council of Finland via grant 370538.
*Taku Onodera*: supported by Research Council of Finland via grant 323233.
*Leena Salmela*: supported by Research Council of Finland via grants 323233 and 370538.

## 1 Introduction

Short-read assemblers [32, 2] have historically relied on *de Bruijn graphs* (DBGs) [14, 27]. These methods decompose reads into $k$-mers, construct the corresponding DBG, and reconstruct the genome by traversing maximal unary paths. The approach is effective because small $k$ yields compact graphs that are cheap to build and, for short reads, recover most of the genome. Nevertheless, finding a value of $k$ (the DBG order) that maximizes assembly quality is not trivial [8]: small $k$ values produce tangled graphs, whereas large $k$ values cause fragmentation when genome complexity or sequencing coverage varies.



© Diego Díaz-Domínguez, Pierfrancesco Martinello, Taku Onodera, Simon J. Puglisi, and Leena Salmela;
licensed under Creative Commons License CC-BY 4.0





Advances in DNA sequencing have produced longer and more accurate reads. PacBio HiFi reads, for example, are typically 10–20 kbp long with error rates below 1% [31]. Standard DBG-based strategies are less suitable in this case because they require large values of $k$ to disentangle the graph, which increases its size and computational cost. Moreover, a large fixed $k$ does not fully exploit the information in HiFi data.

Modern assemblers typically prefer the overlap–layout–consensus (OLC) [21] strategy for long reads. This method builds a graph where the nodes represent the reads and the edges model suffix-prefix overlaps between them. However, OLC requires the costly computation of the overlaps, a task further complicated by homopolymer-length errors, the dominant error type in PacBio HiFi reads [31]. Tools based on the OLC paradigm include Peregrine [9], HiCanu [23], and hifiasm [6]. Although the recent development of fast heuristics for approximate pattern matching has reduced the costs of computing overlaps [3, 25, 18], obtaining OLC graphs from HiFi reads remains expensive. This has motivated a renewed interest in DBG-based approaches that scale more gracefully.

The main challenge is how to combine the information across multiple DBG orders. Some methods progressively increase $k$, as in IDBA [26] and SPAdes [2], whereas others selectively increase $k$ in complex regions, such as LJA [1]. A recent method, DVOUG [20], takes the opposite approach, constructing an initial assembly with a large $k$ value and then extending contigs by progressively lowering $k$. Other strategies reduce graph size through sparsification or compression: MBG [29] constructs a sparse DBG from a subset of $k$-mers, while mdBG [12] performs assembly in minimizer space. More generally, models supporting variable-order contexts have been proposed. Manifold DBGs [19] allow arbitrary substrings as node labels, and variable-order DBGs (voDBGs) [4] represent all DBGs up to maximum order $K$.

Among these alternatives, voDBGs provide an appealing framework for long-read assembly as they simultaneously encode multiple DBG orders and are closely related to compressed suffix trees [22]. Such structures admit efficient construction [11, 24] and compact representation [5], suggesting that voDBGs can be built and stored efficiently as well.

However, a fundamental theoretical gap remains. In fixed-order DBGs, contigs are defined by node in- and out-degrees, and following an edge advances sequence context. In a voDBG, edges represent context change, but also order change[1], and contig definitions such as unitigs and omnitigs [30] do not account for the latter, nor do they leverage the $k$-mer frequencies, which voDBGs naturally encode and which can inform the assembly. There is no formal contig formulation in voDBGs that integrates the structure of multiple DBG orders with $k$-mer frequency.

### Our contribution

We present the first formal framework to enumerate contigs from a voDBG. Our method takes as input a range $[\ell, h]$ bounding the frequency of sequenced substrings across the genome and restricts the voDBG to nodes whose frequency in the reads lies within $[\ell, h]$.

We combine the frequency restriction with a traversal that visits DBG order edges, varying the context length $k$ along the walk. Building a contig in one direction, the traversal right-extends – increasing $k$ – while the node frequency (i.e., number of occurrences of the associated $k$-mer in the reads) stays within $[\ell, h]$. When the frequency drops below $\ell$, it left-contracts – decreasing $k$ – until the range is recovered, then resumes extending. The mirror operations (left extension, right contraction) build the contig in the opposite direction. This procedure implicitly traces a walk over the layout of overlapping reads without ever building one, placing our approach between DBGs and OLC.

[1] Moving from a node with context length $k$ to another with context length $k' \neq k$

We show that requiring $\ell > h/2$ forces every node in the restricted voDBG to have at most one incoming and one outgoing extension edge (and likewise for contraction edges). This lets us partition the restricted voDBG into paths whose labels spell genome segments under uniform coverage and an error-free model. We refer to the path labels as $(\ell, h)$-tigs.

We analyze how the choice of $\ell$ and $h$ affects the contiguity and accuracy of the assembly and provide a method for choosing appropriate values of $\ell$ and $h$ under the same assumptions. We further exploit connections between the voDBG and compressed indexes to develop an efficient, homopolymer-aware algorithm for enumerating $(\ell, h)$-tigs. We implement our ideas in a tool called `Ryu` and show that our approach achieves better contiguity relative to unitigs in fixed-order DBGs. Full-featured assemblers (`Flye` and `Hifiasm`) still achieve higher contiguity through orthogonal steps – iterative graph cleaning, repeat resolution, and polishing – layered on top of contig enumeration. `Ryu` does not perform such post-processing, yet achieves comparable accuracy at lower computational cost, suggesting that $(\ell, h)$-tigs are a strong foundation for an assembly pipeline.

# 2 Preliminaries

## 2.1 DNA strings

A *read* $R[1..n]$ is a sequence over the alphabet $\Sigma = \{\mathtt{a}, \mathtt{c}, \mathtt{g}, \mathtt{t}\}$. Standard notions of suffix, prefix, and substring apply. A *run* is a maximal substring of identical symbols. A run of length greater than one is called a *homopolymer*. The run-length encoding of $R = c_1^{l_1} c_2^{l_2} \cdots c_r^{l_r}$ is $rle(R) = (c_1, l_1), (c_2, l_2), \ldots, (c_r, l_r)$.

The *complement* $c : \Sigma \to \Sigma$ maps $\mathtt{a} \leftrightarrow \mathtt{t}$ and $\mathtt{c} \leftrightarrow \mathtt{g}$; the *reverse complement* of $R$ is denoted $\overline{R}$. A *read collection* is a multiset $\mathcal{R} = \{R_1, R_2, \ldots, R_q\}$ of DNA strings. Its extended version $\mathcal{R}_{ext} = \{R_1, \overline{R_1}, R_2, \overline{R_2}, \ldots, R_q, \overline{R_q}\}$ contains each read together with its reverse complement. We denote by $\rho$ the maximum read length.

## 2.2 de Bruijn graphs

The node-centric *de Bruijn graph* (DBG) of order $k$ [10] is a labeled graph $G = (V, E)$ in which each node $u \in V$ represents a distinct $k$-mer $label(u) \in \Sigma^k$. There is an edge $(u, v) \in E$ labeled $label(v)[k]$ iff $label(u)[2..k] = label(v)[1..k-1]$.

Given a read collection $\mathcal{R}$, the induced graph $G^{\mathcal{R}} \subseteq G$ contains only those nodes whose labels occur in $\mathcal{R}$ and those edges $(u, v)$ such that $label(u) \cdot label(v)[k]$ occurs in $\mathcal{R}$. We define the operator $f(u)$ to denote the frequency of $label(u)$ in $\mathcal{R}$.

## 2.3 Variable-order DBGs

The *variable-order de Bruijn graph* (voDBG) [4] of $\mathcal{R}$ combines the DBGs of all orders $1 \le k \le \rho$, where $\rho$ is the maximum read length. Its node set consists of all distinct substrings of $\mathcal{R}$, where the nodes with a given length $k$ are the nodes of a DBG of order $k$.

For a node $u$ of order $k$:

**Right-extension edge:** $(u, v)$ to a node $v$ of order $k+1$ exists if $label(u) = label(v)[1..k]$. The edge is labeled with $label(v)[k+1]$.

**Left-contraction edge:** $(u, w)$ to a node $w$ of order $k-1$ exists if $label(u)[2..k] = label(w)$.

Left extension and right contraction edges are defined symmetrically.

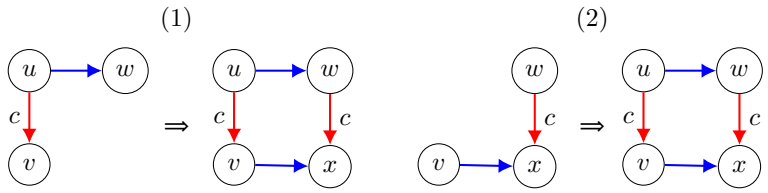



**Figure 1** Lemma 3. Red edges are extensions and blue edges are contractions. Extension and contraction edges commute.

## 3 Our framework

Our assembly method traverses the voDBG of the reads along paths that alternate extension and contraction edges. Extensions refine sequence context and continue while possible; contractions restore context when extensions stop. A contig is formed by the label of the starting node followed by the labels of the visited extension edges.

Extensions decrease frequency and contractions increase it, so we restrict the traversal to nodes whose frequencies lie within the sequencing-coverage interval.

### 3.1 Description of contigs in the voDBG

In this section, we present our formal concept of contigs in a voDBG $G = (V, E)$ induced from a read collection $\mathcal{R}$.

▶ **Definition 1** (Frequency-restricted voDBG). *Given an interval $[\ell, h]$, a frequency-restricted voDBG $G_{\ell,h} \subseteq G$ retains each node $u$ such that $\ell \le f(u) \le h$ and each edge $(u, v)$ such that $\ell \le f(u) \le h$ and $\ell \le f(v) \le h$.*

▶ **Lemma 2.** *When $h/2 < \ell < h$, each node in $G_{\ell,h}$ has at most one outgoing and at most one incoming extension edge. Similarly, each node has at most one outgoing and at most one incoming contraction edge.*

**Proof.** Suppose a node $v \in V$ of order $k+1$ has two incoming extension edges $(u_1, v)$ and $(u_2, v)$. Both $u_1$ and $u_2$ are the prefixes $label(v)[1..k] = label(u_1) = label(u_2)$ and therefore $u_1 = u_2$. Thus, $v$ has one incoming extension edge.

Now suppose a node $u \in V$ of order $k$ has two outgoing extension edges $(u, v_1)$ and $(u, v_2)$ with $label(v_1)[k+1] \neq label(v_2)[k+1]$. Since $label(v_1)[1..k] = label(v_2)[1..k] = label(u)$, it follows that $f(u) \ge f(v_1) + f(v_2)$.

Because $v_1, v_2 \in V$, we have $f(v_1), f(v_2) \ge \ell$, and therefore $f(u) \ge 2\ell$. Since $\ell > h/2$, we also obtain $2\ell > h$, and hence $f(u) \ge 2\ell > h$. This inequality contradicts $u \in V$, which requires $f(u) \le h$. Therefore, $u$ has at most one outgoing extension edge.

The statement for contraction edges follows by symmetry. ◀

Notice the condition $\ell > h/2$ is tight: if $\ell \le h/2$, branching nodes may exist. From now on, we will assume $\ell > h/2$ holds in $G_{\ell,h}$.

▶ **Lemma 3.** *Extension and contraction edges in $G_{\ell,h}$ commute: (1) if $(u, v)$ is an extension edge labeled $c$ and $(u, w)$ is a contraction edge, there exists a node $x$ such that $(v, x)$ is a contraction edge and $(w, x)$ is an extension edge labeled $c$ and (2) if $(w, x)$ is an extension edge labeled $c$ and $(v, x)$ is a contraction edge, there exists a node $u$ such that $(u, w)$ is a contraction edge and $(u, v)$ is an extension edge labeled $c$. See Figure 1.*

**Proof.** (1) Let $w, u$ and $v$ be nodes with orders $k-1$, $k$ and $k+1$, respectively, such that the extension $(u, v)$ and the contraction $(u, w)$ exist. Then it follows $label(v) = label(u)\cdot c$ and $label(w) = label(u)[2..k]$.

Define a node $x$ with $label(x) = label(w)\cdot c = label(u)[2..k]\cdot c$. This implies that $(w, x)$ is an extension edge labeled $c$. Furthermore, since $label(v) = label(u)\cdot c$, a left contraction yields $label(u)[2..k]\cdot c = label(x)$, so $(v, x)$ is a contraction edge.

An extension decreases frequency and a contraction increases it, so we have $f(v) \le f(x) \le f(w)$. Because $u, v, w \in V$ and their frequencies lie in $[\ell, h]$, it follows that $f(x) \in [\ell, h]$, hence $x \in V$.

(2) follows by symmetry. ◀

Because contraction edges are non-branching (Lemma 2), they partition $G_{\ell,h}$ into maximal directed paths, which we call *contraction paths*.

▶ **Lemma 4.** *Let $U = (u_i)_{1\le i\le n}$ and $P = (p_i)_{1\le i\le m}$ be distinct contraction paths in $G_{\ell,h}$. (1) If there is an extension edge $(u_i, p_1)$ with $1 \le i \le n$ labeled with $c$, then for every $0 \le j \le n-i$, there is also an extension edge $(u_{i+j}, p_{1+j})$ labeled $c$. (2) If there is an extension edge $(u_n, p_i)$ with $1 \le i \le m$ labeled with $c$, then for every $0 \le j < i$, there is also an extension edge $(u_{n-j}, p_{i-j})$ labeled $c$. See Figure 2.*

**Proof.** (1) By assumption when $j = 0$, the extension edge $(u_{i+j}, p_{1+j})$ labeled $c$ exists. If $1 \le j \le n-i$ and the extension edge $(u_{i+j-1}, p_{1+j-1})$ labeled $c$ exists, then by Lemma 3 also the extension edge $(u_{i+j}, p_{1+j})$ labeled $c$ exists. Thus by induction the extension edge $(u_{i+j}, p_{1+j})$ labeled $c$ exists for all $0 \le j \le n-i$.

(2) follows by symmetry. ◀

Lemma 4 allows us to lift $G_{\ell,h}$ to a meta-graph:

▶ **Definition 5** (voDBG meta-graph). *The meta-graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ of $G_{\ell,h}$ is defined as follows:*

*Each maximal contraction path $U$ in $G_{\ell,h}$ corresponds to a node in $\mathcal{V}$. Since contraction strictly decreases the order, $U$ contains a unique node of maximal order. The sequence $label(U)$ is defined as the label of that node.*

*Two meta-graph nodes $U, P \in \mathcal{V}$ are connected by an edge $(U, P) \in \mathcal{E}$ labeled $c$ iff there exists an extension edge labeled $c$ in $G_{\ell,h}$ between $U$ and $P$.*

By Lemma 4, if a node $u_i \in U$ has an extension edge labeled $c$ to a node in contraction path $P$, all successive nodes in $U$ also have an extension edge labeled $c$ to a node in contraction path $P$. By Lemma 2, the outdegree of these nodes is at most one and thus the outdegree of each node in $\mathcal{G}$ is at most one. Similarly, by Lemmas 4 and 2, the indegree of each node in $\mathcal{G}$ is at most one. Consequently, connected components of $\mathcal{G}$ are directed paths or cycles. We use this structure to define $(\ell, h)$-tigs analogously to unitigs in fixed-order DBGs:

▶ **Definition 6** ($(\ell, h)$-tigs). *Let $\mathcal{P} = (U_1, U_2, \ldots, U_q)$ be the nodes of a connected component of $\mathcal{G}$, ordered so that $(U_i, U_{i+1}) \in \mathcal{E}$ for $i \in [1, q-1]$. If $\mathcal{P}$ is a directed cycle, the choice of $U_1$ is arbitrary and $(U_q, U_1) \in \mathcal{E}$. If $\mathcal{P}$ is a directed path, $U_1$ has indegree 0 and $U_q$ has outdegree 0. An $(\ell, h)$-tig is the string formed by $label(U_1)$ followed by the labels of the edges $(U_i, U_{i+1})$ in $\mathcal{P}$.*

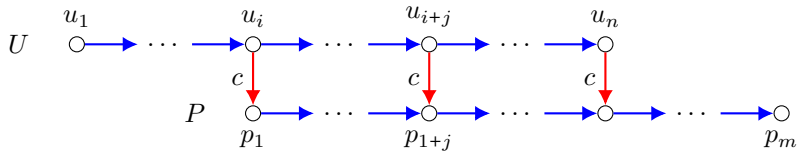



**Figure 2** Lemma 4 (1). $U$ and $P$ are distinct contraction paths in $G_{\ell,h}$. If there is an extension edge from some node $u_i \in U$ to $p_1$, then Lemma 3 implies that all nodes following $u_i$ in $U$ also have an extension edge to a node in $P$.

### 3.1.1 Enumerating $(\ell, h)$-tigs

We spell $(\ell, h)$-tigs by traversing $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. For each $U \in \mathcal{V}$ with indegree 0, we initialize $L = \mathit{label}(U)$ and follow successive extension edges $(U_i, U_{i+1}) \in \mathcal{E}$, appending their labels to $L$, until reaching a node with outdegree 0. The resulting $L$ is an $(\ell, h)$-tig. For components that are directed cycles, we start from an arbitrary node and traverse the cycle once. After visiting all connected components of $\mathcal{G}$, the algorithm outputs one string per component.

### 3.1.2 Connection with genome assembly

We now describe how our $(\ell, h)$-tig enumerating algorithm from Section 3.1.1 relates to genome assembly. Without loss of generality, assume the assembled genome is a single sequence $W[1..w]$, and that the reads in $\mathcal{R}$ come from the same DNA strand.

▶ **Definition 7** (Ground-truth read layout). *For a read $R_i \in \mathcal{R}$, let $[a_i, b_i]$ be the interval from which it originates (i.e., $R_i = W[a_i..b_i]$). The ground-truth read layout is the multiset $\{[a_i, b_i]\}_{i=1}^{q}$ ordered by start position $a_i$. The layout determines, for each position $W[i]$, the reads that cover it.*

Two reads $R_z$ and $R_x$, with $a_z \le a_x$ overlap iff $a_x < b_z$, in which case they share $o = b_z - a_x + 1$ symbols and their collapse $R_z[1..n_z]R_x[o+1..n_z]$ equals $W[a_z..b_x]$. Thus, any genome substring is reconstructed by taking the reads that cover it and collapsing their sequences according to their overlaps in the layout. Consequently, any interval of $W$ covered by a chain of pairwise-overlapping reads equals the collapse of that chain. Reconstructing $W$ without knowing the read layout is the assembly problem.

The $(\ell, h)$-tig enumeration algorithm performs an approximate traversal of the ground-truth layout. Specifically, walking over the nodes of a component in the meta-graph $\mathcal{G}$ traverses the layout symbol by symbol along $W$. This traversal is approximate for two reasons. First, each component node $U_i$ identifies all occurrences of a string $\mathit{label}(U_i)$ regardless of their genomic origin, conflating distinct loci such as repeats. Second, the range $[\ell, h]$ constrains $k$-mer frequency, which only approximates the positional coverage of the layout.

**Comparison with other assembly models.** Our approach sits between the DBG and OLC models and draws on the strengths of both. From DBGs, it inherits an overlap-free construction: the voDBG is built directly from the $k$-mers, with no all-against-all overlap computation. From OLC, it inherits the ability to work across multiple DBG orders, extending and contracting the context dynamically as the layout traversal proceeds. This allows the assembly to extend contigs where fixed-order DBGs would fragment them.

### 3.1.3 Correctness of the reconstruction

Let us consider the conditions under which the $(\ell, h)$-tig enumeration algorithm of Section 3.1.1 reconstructs a genome – that is, a single component of $\mathcal{G}$ spells an entire chromosome. We prove the single-chromosome case, assuming $W[1..w]$ is linear and haploid, and write $f(S)$ for the frequency of a string $S$ in $\mathcal{R}$. The case when $W$ is circular is similar.

For the algorithm's approximate layout traversal to reconstruct $W$, two requirements must hold. First, the occurrences counted by $f(label(U_i))$ must all come from reads overlapping at a single locus; otherwise $label(U_i)$ conflates distinct genomic regions, and its frequency no longer reflects local coverage. Second, every position of $W$ must be covered by some node reachable by the traversal, so that no part of $W$ is skipped. The following theorem makes these requirements precise as frequency conditions on the substrings of $W$.

▶ **Theorem 8** (Full genome reconstruction). *The $(\ell, h)$-tig algorithm of Section 3.1.1 reconstructs $W$ from $\mathcal{R}$ if the following conditions hold:*

1. *If $S \in \Sigma^*$ is a repeat in $W$, then $f(S) > h$.*
2. *The longest prefix $W[1..i_p]$ such that $f(W[1..i_p]) \in [\ell, h]$ exists.*
3. *For each $i$, $i_p \le i < w$, there exists a $j \le i$ such that*
   a. *$f(W[j..i]) \in [\ell, h]$ and*
   b. *$f(W[j..i+1]) \in [\ell, h]$.*

**Proof.** Condition 1 ensures that the node labels of $G_{\ell,h}$ are unique in $W$. Condition 2 ensures that $W[1..i_p]$ is a node of $G_{\ell,h}$: its frequency is at most $h$, so by Condition 1 it is not a repeat in $W$. Because $W[1..i_p]$ is not a repeat and is the longest prefix of $W$ whose frequency is within $[\ell, h]$, it labels a node $U_1$ in $\mathcal{G}$ with indegree 0 from which the algorithm starts extending an $(\ell, h)$-tig. Condition 3 ensures that a string $label(U_i)$ with frequency in $[\ell, h]$ extends by one symbol to a string $label(U_{i+1})$ whose frequency is also in $[\ell, h]$; iterating from $U_1$ yields a node chain $U_1, U_2, \ldots, U_q$ ending in a node $U_q$ of outdegree 0, with $label(U_q)$ a suffix of $W$. By Lemmas 2, 3, and 4, these nodes are connected by extension edges into a unary path in $\mathcal{G}$ labeled $W$ (i.e., the $(\ell, h)$-tig). ◀

A read collection where each repeat of the underlying genome is fully contained in more than $h$ reads fulfills Condition 1. Conditions 2 and 3 require that the coverage of $\mathcal{R}$ is not too low at any position and that there are no positions (except for the first and last positions) on the genome where many reads start or end (i.e., read endpoints are not clustered).

Real data never meets the conditions of Theorem 8 in practice. We nonetheless analyze assembly under this scenario first because it isolates the structural reason $(\ell, h)$-tigs reconstruct the genome. In the next section, we will discuss how to choose $[\ell, h]$ from $\mathcal{R}$ to balance contiguity against the risk of misassembly when the assumptions fail.

### 3.1.4 Modeling misassemblies

In practice, reads are affected by sequencing errors, unsolved repeats, non-uniform coverage, and allele-specific coverage variation. These factors cause $(\ell, h)$-tigs to be interrupted (fragmentation) or deviate from true genomic substrings (misassembly).

*Fragmentation* occurs when the true genomic continuation of a $(\ell, h)$-tig is removed from $G_{\ell,h}$ because its coverage drops below $\ell$.

*Misassemblies* arise when the frequency of a repeat of the genome is within the range $[\ell, h]$ and the number of reads supporting the correct extension decreases and matching reads from other contexts keep their frequency within $[\ell, h]$. This problem combines a lack of coverage with spurious overlaps generated by sequencing errors and/or repeats. Figure 3 shows this scenario.

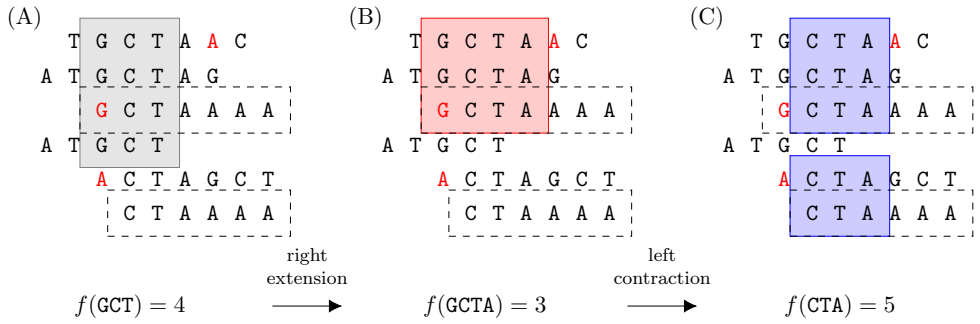



**Figure 3** Genome assembly as a sequence of extensions and contractions with $\ell = 3$ and $h = 5$. Overlapping reads represent the ground-truth layout. The dashed reads are misplaced in the layout as they come from other genomic regions but form spurious matches with the correctly placed reads. Red symbols indicate sequencing errors. The colored boxes depict the sequence context the $(\ell, h)$-tig algorithm can see at every step. In (A), `GCT` loses one occurrence due to a mismatch in `ACTAGCT`, and gains an occurrence from the spurious overlap with `GCTAAAA`. In (A-B), the right-extension `GCT` → `GCTA` has $f(\texttt{GCTA}) = 3$. Two of its occurrences come from the correct reads in the layout, while the third comes from a spurious overlap. In (C), `CTA` would be incorrectly extended with `A` because of the spurious overlap created by the mismatch in `TGCTAAC`.

When the reads in $\mathcal{R}$ do not satisfy the ideal conditions of Theorem 8, the interval $[\ell, h]$ must balance misassembly and fragmentation. We show how to choose $\ell$ and $h$ to control the probability of these events.

### 3.1.5 Fragmentation

Let $\delta = h - \ell + 1$ be the smallest frequency change that pushes a node out of $[\ell, h]$, disrupting the extension of an $(\ell, h)$-tig. Connectivity is broken when a contraction or extension changes frequency by at least $\delta$.

Consider a substring $S = W[i..i + k - 1]$. If $\geq \delta$ reads start at $i$, then contraction from $cS = W[i - 1..i + k - 1]$ yields $f(S) \geq f(cS) + \delta > h$, deleting the edge from $cS$ to $S$. Symmetrically, if $\geq \delta$ reads end at $i + k - 1$, the extension $Sc = W[i..i + k]$ yields $f(Sc) \leq f(S) - \delta < \ell$, deleting the edge from $S$ to $Sc$.

Sequencing errors transfer occurrences of $S$ in $\mathcal{R}$ to other strings $S'$, decreasing $f(S)$ and increasing $f(S')$. Given an error rate $e$, coverage $C$, and genome size $w$, the expected number of such transfers is $eCw$.

Assuming these events – sequencing errors and read endpoints – occur uniformly across the genome, the maximum number of events at a single position bounds the maximum perturbation of a $k$-mer's frequency. The problem of choosing the values of $(\ell, h)$ therefore reduces to a balls-into-bins model with $|\mathcal{R}| + eCw$ balls and $w$ bins. By Raab and Steger [28], if

$$\delta \geq 2\frac{\ln w}{\ln \frac{w \ln w}{|\mathcal{R}| + eCw}}, \tag{1}$$

then with high probability[2] no position of $W$ is the start or end of $\geq \delta$ reads. Here ln denotes the natural logarithm.

[2] A high probability here is a probability $1 - o(1)$, i.e., the probability tends to 1 as $w$ grows to infinity.

### 3.1.6 Misassemblies

Suppose $S$ has two genomic extensions $Sc_1$ and $Sc_2$. Under uniform coverage their read counts follow a binomial distribution with parameter 0.5. A misassembly occurs if one extension has frequency $i \in [\ell, h]$ (kept) while the other $f(S) - i < \ell$ (discarded). The probability of this happening is $P \leq 2\sum_{i=\ell}^{h} \binom{h}{i} 0.5^i 0.5^{h-i} = 2^{1-h} \sum_{i=\ell}^{h} \binom{h}{i}$.

Let $d = \ell - h/2$ denote the deviation from the mean. By the Chernoff bound, $P \leq 2e^{-2d^2/h}$. If a misassembly probability at most $\epsilon$ is tolerated, then

$$\ell \geq \frac{h}{2} + \sqrt{\frac{h}{2} \ln\left(\frac{2}{\epsilon}\right)}. \tag{2}$$

This analysis applies to the two-extension case. Combining Equations 1 and 2, we require

$$\frac{h}{2} + \sqrt{\frac{h}{2} \ln\left(\frac{2}{\epsilon}\right)} \leq \ell \leq h - 1 - 2\frac{\ln w}{\ln \frac{w \ln w}{|\mathcal{R}| + eCw}}. \tag{3}$$

We choose the smallest $h$ for which an integer $\ell$ satisfies Equation 3.

**Scope and limitations.** It is worth noting that the interval $[\ell, h]$ of Equation 3 neither minimizes fragmentation nor maximizes contiguity. Rather, it gives values of $\ell$ and $h$ sufficient to keep the per-node misassembly probability below $\epsilon$. The analysis remains idealized, as it assumes frequency fluctuations are distributed uniformly across $W$; non-uniform and allele-specific coverage break that assumption. Its value is that, even in this simplified setting, sequencing errors are budgeted explicitly into the fluctuation bound $\delta$ – unlike the standard fixed-order DBG formulation, where the choice of $k$ does not account for frequency or error rate. In order to compensate for the lack of both uniform total and local coverage in empirical reads, we later define a way to calculate a recommended set of parameters (see Section 5.1).

## 4 Practical genome assembler

We present a practical long-read assembler based on $(\ell, h)$-tigs. Because sequencing samples both DNA strands, the input is the extended collection $\mathcal{R}_{ext}$ containing each read and its reverse complement. We assume suitable values of $(\ell, h)$ (Section 3.1.6) computed from $\mathcal{R}$ and doubled for $\mathcal{R}_{ext}$.

Long-read technologies often misestimate homopolymer lengths, producing spurious overlaps and potential misassemblies. We incorporate a simple heuristic to mitigate this issue.

### 4.1 Preparing the data

Similar to previous work [23, 29, 1], our homopolymer-error heuristic keeps each read $R \in \mathcal{R}_{ext}$ in run-length format $rle(R) = (c_1, l_1)(c_2, l_2), \ldots, (c_r, l_r)$. We separate the compressed read into its *symbol sequence* $c_1, c_2, \ldots, c_r$ and *length sequence* $l_1, l_2, \ldots, l_r$. Let $\mathcal{R}_c$ be the collection of symbol sequences.

We index $\mathcal{R}_c$ using a compressed structure $I$ with suffix-tree functionality (see Section 5) that supports navigational queries in the voDBG $G_{\ell,h} = (V, E)$ of $\mathcal{R}_c$. The length sequences are integrated into $I$ so that for each node $u \in V$ we can query

- $hp(u)$: return the list of length sequences corresponding to the uncompressed occurrences of $label(u)$ in $\mathcal{R}_{ext}$.

The index $I$ maps all occurrences of a substring $S$ in $\mathcal{R}_{ext}$ to the same node $u$ in $G_{\ell,h}$ based on their symbol sequence, ignoring homopolymer length differences. This removes spurious overlaps caused by homopolymer errors. However, different strings $S \neq S'$ with the same symbol sequence may also collapse to the same node. The lists returned by $hp(u)$ allow us to detect such cases: homopolymer errors produce small deviations around the true lengths, whereas multiple strings typically generate distinct length distributions.

Because reverse complements are included, the meta-graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ contains two components per chromosome, one for each DNA strand. For a node $U \in \mathcal{V}$ we denote by $\overline{U}$ the node labeled with the reverse complement of $label(U)$. We also define $hp(U) = hp(u_1)$, where $u_1$ is the highest-order node of $U$.

## 4.2 Assembly

The algorithm selects a pair of nodes $U, \overline{U} \in \mathcal{V}$ with indegree and outdegree 0, respectively. These correspond to the starting and ending points of complementary components of $\mathcal{G}$.

It reconstructs an uncompressed sequence for $S = label(U)$ using the list $\mathcal{L} = hp(U)$. For each position $i$ in $S$, the homopolymer length is estimated as the median of the lengths at position $i$ in $\mathcal{L}$. If the standard deviation exceeds a threshold $\tau$, the length at position $i$ is marked as *fuzzy*.

After processing all positions, if the fraction of fuzzy lengths exceeds a threshold $\mu$, $(U, \overline{U})$ is discarded and the algorithm moves to another component of $\mathcal{G}$. Otherwise, the algorithm spells the $(\ell, h)$-tig starting at $U$. The uncompressed length of each extension label $c$ is estimated in the same way using the median.

Once the $(\ell, h)$-tig is completed, the nodes of $U$ and $\overline{U}$ are marked as processed, and the algorithm continues with the next unvisited pair $(U, \overline{U})$.

# 5 Experiments

## 5.1 Setup

**Implementation details.** We implemented our $(\ell, h)$-tig assembler (Section 4) in `C++` as `Ryu`[3]. The tool builds a compressed FMD-index [17] over the symbol sequences $\mathcal{R}_c$. The index uses the bidirectional BWT [16], computed with `grlBWT` [11]. Traversing BWT ranges corresponding to suffix tree nodes enumerates voDBG nodes $u$ and their frequencies. Run-length encoding lengths from $\mathcal{R}_{ext}$ are stored in BWT order to support $hp(u)$ queries, enabling homopolymer reconstruction. `Ryu` then visits the connected components of the meta-graph $\mathcal{G}$ via BWT backward search and spells $(\ell, h)$-tigs as described in Section 4.

`Ryu` currently supports limited parallelization, using up to four threads. To distribute work among them, the FMD-index is divided into ranges, which are then assigned to threads to compute the $(\ell, h)$-tigs in parallel. However, without first processing the FMD-index, it is difficult to ensure that each thread receives a similar workload. Therefore, we adopt a simpler heuristic that partitions the data structure according to the DNA alphabet.

**Datasets.** We used three PacBio HiFi read collections: *E. Coli* (ECOLI), *S. Cerevisiae* (YEAST), and the human cell line CHM13 (HUMAN). While ECOLI and HUMAN are complete genomes, the YEAST genome is at a contig level. We assembled these datasets with different

[3] `https://github.com/ddiazdom/Ryu`, commit `0eb064a`

methods and evaluated the output using `Quast` [13]. Table 1 lists the SRA accessions of the reads and the reference genomes used for evaluation, alongside the genome sizes, and reads coverage.

**Competitor tools.** We compared `Ryu` with `Bcalm2`[4] [7], `Flye`[5] [15], and `Hifiasm`[6] [6] for assembly. `Bcalm2` outputs unitigs from a fixed-order DBG, `Flye` constructs A-Bruijn and repeat graphs, and `Hifiasm` follows an OLC strategy based on minimizer-detected overlaps. `Bcalm2` and `Ryu` only prune the graph based on $k$-mer frequency and spell sequences from the pruned graph, whereas `Hifiasm` and `Flye` perform full genome assembly. The YEAST reference genome has been assembled using `Flye v.2024.0.7`, so the results may be biased for this particular dataset.

**Experimental setup.** For all the compared tools, a set of common parameters have been used for all the datasets, while for `Ryu` some particular dataset-specific parameters have been added. For `Ryu`, we varied $\ell$ $(9-24)$ and $h$ $(16-32)$. While assembling in parallel, in order to avoid contention in fragmented assemblies, we omit $(\ell, h)$-tigs with length smaller than 1000 base pairs using `-r 1000` (see discussion in Section 5.2.3) hence renouncing less than 1% of the total length of the $(\ell, h)$-tigs assembly. For `Bcalm2`, $k$ was varied in the range between $2^4$ and $2^8$, and the abundance threshold (`-abundance-min`) from 7 to 15, for a total of 75 different parameters combinations. `Flye` and `Hifiasm` were run with default parameters. The parameters common to each dataset are shown in detail in Table A1.

For each dataset, we calculated the theoretical optimal values of $(\ell, h)$ for each dataset (Section 3.1.6) with confidence rate 0.95, and the recommended range $[\ell, h]$, on which $h$ is set to half of the coverage and $\ell = \frac{2}{3}h$. The coverage has been calculated by our tool `Ryu` as the rate between the number of symbols in the index $I$ and the expected size given as input. In our experiments, setting the interval $[\ell, h]$ this way produced good assemblies for all datasets. The experiments were executed with 24 threads to allow a comparison of memory usage and time between the tools. The only exception was `Ryu`, which used up to 4 threads due to implementation limitations.

Figure 4 illustrates the tradeoffs between assembly quality and the choice of the values $(\ell, h)$ (exact values are reported in Table A1). Assembly statistics for the compared tools are summarized in Table 2, while peak memory usage and runtime are listed in Table 3. For `Bcalm2`, we report the configuration with the fewest misassemblies in each dataset. For `Ryu`, we highlight three configurations: *best assembly* (`Ryu` ★), which achieves the highest N50 and genome coverage, *theoretical best* (`Ryu`▲), with $[\ell, h]$ calculated according to Equation 3, and *best performance* (`Ryu` ◆), which completes the assembly of reads in the shortest time. `Ryu` ◆ coincides with the recommended parameters.

**Machine specifications.** We performed the experiments in a cluster environment running AlmaLinux 8.10 with kernel Linux 4.18.0, using a single node per execution: the node uses an Intel(R) Xeon(R) Gold 6248 CPU, with 80 threads and a maximum frequency of 3.9 GHz, with 380GB RAM. All tools were compiled with `GCC 14.2.0` according to the authors' specifications. We compiled `Ryu` using `-O3`. To achieve speed up on larger values of $k$, `Bcalm2` has been compile by providing a custom `-DKSIZE_LIST`.

[4] `https://github.com/GATB/bcalm`, commit `fe8a8c5`
[5] `https://github.com/mikolmogorov/Flye`, commit `886b8c1`
[6] `https://github.com/chhylp123/hifiasm`, commit `ec9a8b2`

**Table 1** Species and reference genomes used in the experiments. The reference genomes were downloaded from the NCBI database: `https://www.ncbi.nlm.nih.gov/datasets/genome`.

| Scientific name | Dataset | SRA accession | Ref. genome GenBank acc. | Genome size (Mbp) | Seq. coverage |
|---|---|---|---|---|---|
| *E. Coli* | ECOLI | SRR10971019 | GCA_000005845.2 | 4.64 | 300 |
| *S. Cerevisiae* | YEAST | SRR34579673 | GCA_051942865.1 | 11.36 | 676 |
| *H. Sapiens* | HUMAN | SRR11292121 | GCA_009914755.1 | 3,117.28 | 9.57 |

## 5.2 Results and discussion

### 5.2.1 Effect of $\ell$ and $h$ on the assembly

Figure 4 shows how different choices of $\ell$ and $h$ affect the results of the assembly. These experiments reveal a tradeoff between contiguity and correctness controlled by the assembly parameters. Additionally, the different coverage and increasing repeat complexity of the datasets introduces further challenges that influence this tradeoff.

For ECOLI (first row of Figure 4), small values of $\ell$ and $h$ produce a larger number of $(\ell, h)$-tigs, whereas larger values reduce their number. As a consequence, assemblies generated with small $\ell, h$ values are more fragmented and yield smaller N50 values. In contrast, the number of misassemblies follows the opposite trend: smaller values of $\ell$ and $h$ produce fewer misassemblies, while larger values increase them.

This behavior can be explained by the way the range $[\ell, h]$ is used. When $\ell$ and $h$ are small, the first node $u$ from which the extension and an $(\ell, h)$-tig can be extended typically has a long label $label(u)$. This happens because the assembly algorithm requires several extensions before reaching a node $u$ such that $f(u) \in [\ell, h]$. At this stage, no contractions have yet been performed, so the assembly has not introduced misassemblies and the constructed prefix of the $(\ell, h)$-tig is correct. However, the subsequent chain of contractions and extensions starting from $u$ tends to terminate quickly: because the interval $[\ell, h]$ is narrow for small values of $(\ell, h)$ (by Lemma 2), the frequency of the extending $(\ell, h)$-tig soon falls outside the allowed range, which limits further extension and results in shorter contigs.

The situation in YEAST is somewhat different (second row of Figure 4). As in ECOLI, increasing $(\ell, h)$ reduces the number of reconstructed sequences. However, the number of misassemblies is the smallest for the recommended values of $(\ell, h)$. Otherwise the number of misassemblies is minimized for the intermediate choice $\ell = 17, h = 31$. This more complex behaviour likely reflects the higher structural complexity of the dataset, where unsolved repeats in the reads and homopolymer compression introduce spurious overlaps.

The coverage of the HUMAN dataset is much lower than that of ECOLI and YEAST datasets and thus the results are different. The results for HUMAN (third row of Figure 4) show that the number of $(\ell, h)$-tigs first increases when $[\ell, h]$ increase. However, after $\ell$ gets larger than the coverage, 9.57x, the number of $(\ell, h)$-tigs starts to decrease because most of the node labels do not have higher occurrence frequencies in the reads and thus the assembly captures less and less of the genome. The assemblies obtained with larger $(\ell, h)$ values also exhibit smaller N50 values, indicating increased fragmentation due to the low coverage.

The number of misassemblies in HUMAN shows an opposite pattern compared to ECOLI. Here, larger values of $(\ell, h)$ lead to fewer misassemblies. In this regard, the assembler achieves zero misassemblies for $(\ell = 13, h = 22)$, $(\ell = 20, h = 26)$, and $(\ell = 24, h = 32)$, although these assemblies were highly fragmented.

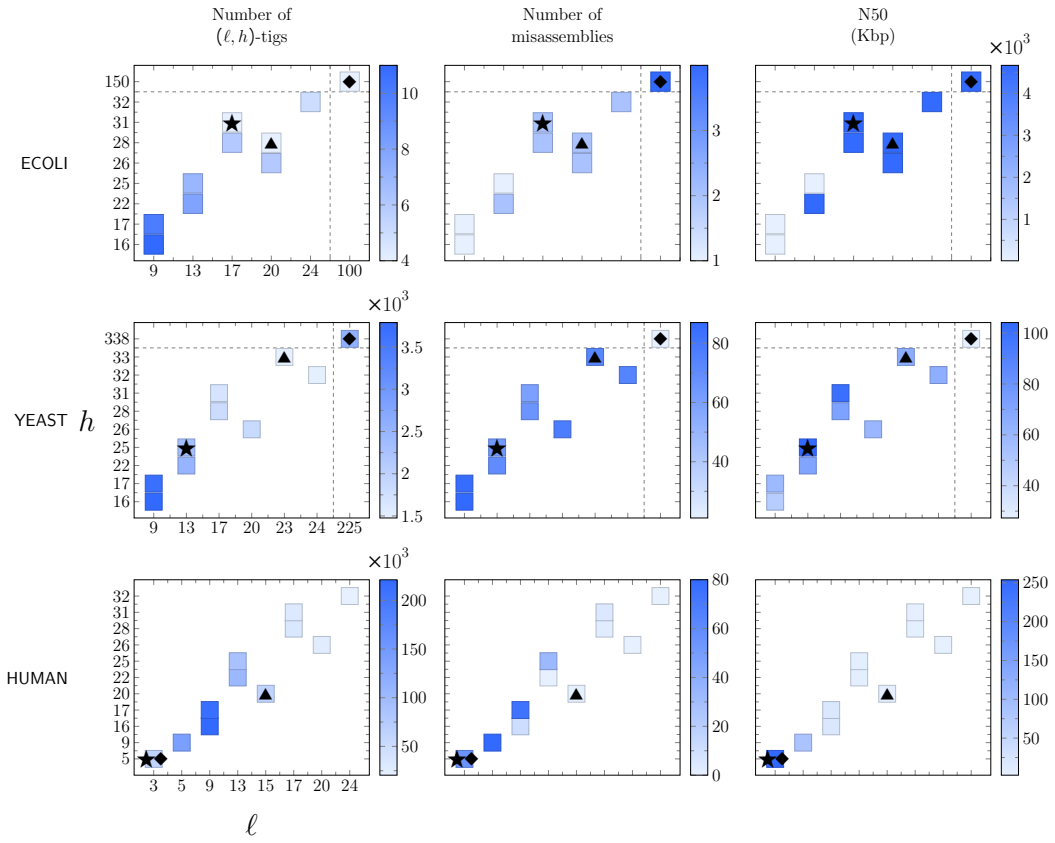



**Figure 4** Effect of $\ell$ and $h$ on the assembly. Each row corresponds to a dataset. The x-axis shows $\ell$ and the y-axis shows $h$. Axes are displayed only in the first column of each row because they are identical across panels. ★, ◆, and ▲ indicate the best, fastest, and theoretical optimum assemblies, respectively. The values of $\ell$ and $h$ (x- and y-axis) do not increase evenly. In particular, the highest $\ell$ and $h$ values on ECOLI and YEAST deviate a lot from the other values and are thus separated by the dashed lines.

The theoretically optimal values for $(\ell, h)$ give a guarantee for avoiding misasemblies with the given probability threshold, while at the same time maximizing contiguity. Thus these values attempt to balance these two effects, and we see that the theoretically optimal values for $[\ell, h]$ do not minimize the number of misassemblies or maximize the contiguity on any of the datasets. On the HUMAN dataset both extremes of the theoretically optimal range, $[\ell, h] = [15, 20]$, are larger than the coverage of the dataset indicating that according to the theory, larger coverage would be needed to guarantee a misassembly probability of 0.95. Still, a reasonable assembly is produced with lower values of $(\ell, h)$, which may be due to the analysis not being tight.

Overall, these results highlight the trade-off inherent in the choice of the pair $(\ell, h)$. Smaller values favour aggressive extension, producing longer sequences but increasing the risk of misassemblies caused by spurious overlaps. Conversely, larger values restrict extension, which reduces misassemblies but leads to more fragmented assemblies. In practice, choosing $h$ to be half the coverage of the dataset and $\ell = \frac{2}{3}h$ provides a good compromise between contiguity and number of misassemblies. However for datasets with high coverage, a better assembly may be achieved with lower $(\ell, h)$ pair because `Ryu` can then take advantage of longer contexts and thus bridge repeats better.

**Table 2** Assembly statistics for `Ryu`, `Bcalm2`, `Hifiasm`, and `Flye`. The misassemblies column shows the number of misassemblies reported by `QUAST`, divided into relocations **R** (flanking sequences align over 1 kbp apart), translocations **T** (flanking sequences align on different chromosomes), and inversions **I** (flanking sequences align on opposite strands). For `Ryu`, ★, ◆, and ▲ indicate the best, fastest, and theoretical optimum assemblies, respectively. The fastest (◆) assembly coincides with the recommended parameters on all datasets.

| Dataset | Tool | Assembly size (Mbp) | N50 (Kbp) | Misassemblies | | | Indels | Total contigs |
|---|---|---|---|---|---|---|---|---|
| | | | | **R** | **T** | **I** | | |
| ECOLI | `Ryu` ★ | 4.67 | 4,665.84 | 1 | 0 | 0 | 5 | 4 |
| | `Ryu` ◆ | 4.65 | 4,640.70 | 1 | 0 | 2 | 3 | 4 |
| | `Ryu` ▲ | 4.64 | 4,640.70 | 1 | 0 | 0 | 3 | 4 |
| | `Bcalm2` | 4.53 | 2.19 | 0 | 0 | 0 | 1 | 2,659 |
| | `Hifiasm` | 4.67 | 4,665.56 | 1 | 0 | 0 | 4 | 1 |
| | `Flye` | 4.64 | 1,182.54 | 1 | 0 | 0 | 3 | 7 |
| YEAST | `Ryu` ★ | 19.27 | 104.14 | 38 | 26 | 0 | 1,925 | 2,322 |
| | `Ryu` ◆ | 12.02 | 47.16 | 7 | 11 | 0 | 457 | 2,522 |
| | `Ryu` ▲ | 23.61 | 65.33 | 47 | 26 | 0 | 1,604 | 1,476 |
| | `Bcalm2` | 10.98 | 1.43 | 0 | 0 | 0 | 355 | 9,753 |
| | `Hifiasm` | 33.27 | 138.67 | 463 | 25 | 0 | 4,334 | 890 |
| | `Flye` | 11.25 | 759.17 | 0 | 8 | 0 | 238 | 823 |
| HUMAN | `Ryu` ★◆ | 3,138.71 | 253.98 | 46 | 8 | 1 | 302,008 | 62,016 |
| | `Ryu` ▲ | 185.98 | 3.92 | 0 | 0 | 0 | 2,601 | 61,922 |
| | `Bcalm2` | 1,249.91 | 1.93 | 0 | 0 | 0 | 157,855 | 827,231 |
| | `Hifiasm` | 2,853.07 | 3,967.22 | 177 | 63 | 1 | 157,855 | 1,453 |
| | `Flye` | 2,789.13 | 5,953.40 | 108 | 128 | 2 | 190,702 | 2,085 |

### 5.2.2 Comparison with other methods

The assembly quality of `Ryu` is comparable to that of long reads assemblers (`Hifiasm`, `Flye`) on ECOLI, and substantially better than unitigs in fixed-order DBG assemblers on HUMAN. On YEAST, the assembly quality of `Ryu` is comparable to that of `Hifiasm`, whereas the assembly quality of `Flye` is better. However, since the used reference genome was also produced with `Flye`, the comparison might be biased towards `Flye`. At the same time, the computational efficiency of `Ryu` remains closer to that of DBG-based tools, highlighting its potential as a lightweight alternative to OLC assemblers. We observe minor differences between the best (★), the theoretical best (▲), and the fastest/recommended (◆) `Ryu` configurations.

Table 2 shows that `Bcalm2` produces a much more fragmented assembly than `Ryu` across all datasets, with very low N50 values and a large number of contigs. By leveraging multiple values of $k$, `Ryu` substantially improves contiguity, producing assemblies with N50 values over 40x higher.

On ECOLI, both `Ryu` and `Hifiasm` reconstruct the genome in a single large contig. `Ryu` produces a few additional short contigs that likely correspond to plasmids or sequencing artifacts, whereas `Flye` splits the genome into several contigs. All methods show similarly low numbers of misassemblies on this dataset.

On the more complex YEAST and HUMAN datasets, `Ryu` produces assemblies that are more contiguous than those of `Bcalm2` but less contiguous than those of `Hifiasm` and `Flye`. However, `Ryu` introduces fewer misassemblies than `Hifiasm` on both datasets and fewer than

**Table 3** Runtime and peak memory usage for the tools used in the experiments. The instance and parameters of `Bcalm2` are the ones yielding the highest N50 value. For `Ryu`, indexing, assembling, and total time are shown for the best (★), fastest (◆), and theoretical optimum (▲) assemblies. The fastest (◆) coincides with the recommended parameters. The values next to `Bcalm2` correspond to $k$ and `-abundance-min`, respectively.

| Dataset | Tool | | | Runtime (sec) | Peak memory (GB) |
|---|---|---|---|---|---|
| ECOLI | `Ryu` | ★ | *indexing* | 176 | 0.63 |
| | | | *assembly* | 210 | 0.09 |
| | | | *total* | 386 | 0.63 |
| | | ◆ | *assembly* | 174 | 0.09 |
| | | | *total* | 350 | 0.63 |
| | | ▲ | *assembly* | 243 | 0.09 |
| | | | *total* | 419 | 0.63 |
| | `Bcalm2` | | (224, 15) | 73 | 6.37 |
| | `Hifiasm` | | | 888 | 16.64 |
| | `Flye` | | | 2,569 | 11.43 |
| YEAST | `Ryu` | ★ | *indexing* | 1,315 | 2.80 |
| | | | *assembly* | 1,550 | 0.42 |
| | | | *total* | 2,865 | 2.80 |
| | | ◆ | *assembly* | 579 | 0.39 |
| | | | *total* | 1,894 | 2.80 |
| | | ▲ | *assembly* | 1,432 | 0.41 |
| | | | *total* | 2,747 | 2.80 |
| | `Bcalm2` | | (224, 13) | 276 | 7.03 |
| | `Hifiasm` | | | 4,974 | 35.62 |
| | `Flye` | | | 21,408 | 28.95 |
| HUMAN | `Ryu` | ★◆ | *indexing* | 7,666 | 12.46 |
| | | | *assembly* | 6,017 | 12.83 |
| | | | *total* | 13,683 | 12.83 |
| | | ▲ | *assembly* | 17,909 | 7.07 |
| | | | *total* | 25,575 | 12.46 |
| | `Bcalm2` | | (40, 7) | 1,580 | 6.92 |
| | `Hifiasm` | | | 3,261 | 49.23 |
| | `Flye` | | | 48,641 | 104.54 |

`Flye` on HUMAN. `Flye` achieves the most contiguous and accurate assembly on YEAST, although this comparison may be biased because the reference genome was also assembled using `Flye`.

In terms of computational efficiency (Table 3), `Bcalm2` is the fastest method. Nevertheless, `Ryu` is substantially faster than `Hifiasm` and `Flye` on ECOLI and YEAST, while remaining faster than `Flye` on HUMAN. This comparison is conservative for `Ryu`, since the other assemblers use 24 threads, whereas `Ryu` currently uses four. Moreover, a significant fraction of the runtime in `Ryu` corresponds to index construction.

Finally, `Ryu` consistently uses less memory than full assemblers, illustrating the low memory usage of DBG-based approaches. `Bcalm2` uses even less memory than `Ryu` on the largest dataset because it builds a fixed-order DBG, but this comes at the cost of a substantially increased fragmentation.

### 5.2.3 A note on parallel efficiency

When `Ryu` runs in multi-threaded mode with values of $\ell$ and $h$ that produce fragmented assemblies, its runtime increases. Parallel execution computes $(\ell, h)$-tigs across multiple threads while maintaining shared state to avoid generating both DNA strands of the same $(\ell, h)$-tig independently. In highly fragmented assemblies, threads spend little time extending the contig and more time synchronizing the shared information, leading to high contention. As a result, multi-threaded execution can paradoxically become slower than single-threaded mode. In our experiments, we dealt with this situation by filtering $(\ell, h)$-tigs shorter than 1000 base pairs: threads discard such contigs and skip the shared-state update. Although this threshold is arbitrary, it had a negligible impact on genome coverage (see Table 2).

## 6 Conclusions and further work

We presented a theoretical framework for assembling long reads using voDBGs. Our experiments show that `Ryu` achieves a strong compromise between contiguity, correctness, and efficiency, significantly improving over fixed-order DBGs while remaining lighter than long-read assemblers. These results suggest that voDBG-based assemblers can offer a practical alternative to expensive OLC methods in long reads.

Further improvements could make our approach a practical full-fledged *de novo* assembler. Advances in read indexing and misassembly detection may improve both efficiency and accuracy, while ongoing progress in algorithms for constructing and indexing large BWTs is likely to enable more scalable implementations. Additionally, dynamically adjusting the interval $[\ell, h]$ based on local read features could help reduce misassemblies, although identifying the most informative signals remains a challenge. On the other hand, scaffolding strategies that exploit the information in the read index may help reduce fragmentation. Finally, extending the framework to polyploid genomes could enable efficient reconstruction of more complex genomes.

## References


1 Anton Bankevich, Andrey Bzikadze, Mikhail Kolmogorov, Dmitry Antipov, and Pavel A. Pevzner. LJA: Assembling long and accurate reads using multiplex de Bruijn graphs. bioRxiv, 2021. doi:10.1101/2020.12.10.420448.

2 Anton Bankevich, Sergey Nurk, Dmitry Antipov, Alexey A. Gurevich, Mikhail Dvorkin, Alexander S. Kulikov, Valery M. Lesin, Sergey I. Nikolenko, Son Pham, Andrey D. Prjibelski, et al. SPAdes: A new genome assembly algorithm and its applications to single-cell sequencing. *Journal of Computational Biology*, 19(5):455–477, 2012. doi:10.1089/cmb.2012.0021.

3 Konstantin Berlin, Sergey Koren, Chen-Shan Chin, James P Drake, Jane M Landolin, and Adam M Phillippy. Assembling large genomes with single-molecule sequencing and locality-sensitive hashing. *Nature biotechnology*, 33(6):623–630, 2015. URL: https://www.nature.com/articles/nbt.3238.

4 Christina Boucher, Alex Bowe, Travis Gagie, Simon J. Puglisi, and Kunihiko Sadakane. Variable-order de Bruijn graphs. In Ali Bilgin, Michael W. Marcellin, Joan Serra-Sagristà, and James A. Storer, editors, *2015 Data Compression Conference, DCC 2015, Snowbird, UT, USA, April 7-9, 2015*, pages 383–392. IEEE, 2015. doi:10.1109/DCC.2015.70.

5 Alexander Bowe, Taku Onodera, Kunihiko Sadakane, and Tetsuo Shibuya. Succinct de Bruijn graphs. In Benjamin J. Raphael and Jijun Tang, editors, *Proc. 12th International Workshop on Algorithms in Bioinformatics (WABI)*, volume 7534 of *Lecture Notes in Computer Science*, pages 225–235. Springer, 2012. doi:10.1007/978-3-642-33122-0_18.

6 Haoyu Cheng, Gregory T Concepcion, Xiaowen Feng, Haowen Zhang, and Heng Li. Haplotype-resolved de novo assembly using phased assembly graphs with hifiasm. *Nature methods*, 18(2):170–175, 2021. doi:10.1038/s41592-020-01056-5.

7 Rayan Chikhi, Antoine Limasset, and Paul Medvedev. Compacting de bruijn graphs from sequencing data quickly and in low memory. *Bioinform.*, 32(12):201–208, 2016. doi:10.1093/bioinformatics/btw279.

8 Rayan Chikhi and Paul Medvedev. Informed and automated k-mer size selection for genome assembly. *Bioinformatics*, 30(1):31–37, 2014. doi:10.1093/bioinformatics/btt310.

9 Chen-Shan Chin and Asif Khalak. Human genome assembly in 100 minutes. bioRxiv, 2019. URL: https://www.biorxiv.org/content/biorxiv/early/2019/07/17/705616.full.pdf.

10 Nicolaas Govert De Bruijn. A combinatorial problem. *Proc. Section of Sciences of the Koninklijke Nederlandse Akademie van Wetenschappen te Amsterdam*, 49(7):758–764, 1946. URL: https://research.tue.nl/files/4442708/597473.pdf.

11 Diego Díaz-Domínguez and Gonzalo Navarro. Efficient construction of the bwt for repetitive text using string compression. *Information and Computation*, 294:105088, 2023. doi:10.1016/j.ic.2023.105088.

12 Barış Ekim, Bonnie Berger, and Rayan Chikhi. Minimizer-space de Bruijn graphs: Whole-genome assembly of long reads in minutes on a personal computer. *Cell Systems*, 12(10):958–968, 2021. doi:10.1016/j.cels.2021.08.009.

13 Alexey Gurevich, Vladislav Saveliev, Nikolay Vyahhi, and Glenn Tesler. QUAST: quality assessment tool for genome assemblies. *Bioinformatics*, 29(8):1072–1075, 2013. doi:10.1093/bioinformatics/btt086.

14 Ramana M. Idury and Michael S. Waterman. A new algorithm for DNA sequence assembly. *Journal of Computational Biology*, 2(2):291–306, 1995. doi:10.1089/cmb.1995.2.291.

15 Mikhail Kolmogorov, Jeffrey Yuan, Yu Lin, and Pavel A Pevzner. Assembly of long, error-prone reads using repeat graphs. *Nature biotechnology*, 37(5):540–546, 2019. doi:10.1038/s41587-019-0072-8.

16 Tak Wah Lam, Ruiqiang Li, Alan Tam, Simon Wong, Edward Wu, and Siu-Ming Yiu. High throughput short read alignment via bi-directional BWT. In *Proc. IEEE international conference on bioinformatics and biomedicine (BIBM)*, pages 31–36. IEEE Computer Society, 2009. doi:10.1109/BIBM.2009.42.

17 Heng Li. Exploring single-sample SNP and INDEL calling with whole-genome de novo assembly. *Bioinformatics*, 28(14):1838–1844, 2012. doi:10.1093/bioinformatics/bts280.

18 Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18):3094–3100, 2018. doi:10.1093/bioinformatics/bty191.

19 Yu Lin and Pavel A. Pevzner. Manifold de Bruijn graphs. In Dan Brown and Burkhard Morgenstern, editors, *Proc. 14th International Workshop on Algorithms in Bioinformatics (WABI)*, Lecture Notes in Computer Science, pages 296–310. Springer, 2014. doi:10.1007/978-3-662-44753-6_22.

20 Zhiqiang Liu, Xue Li andLei Xie, Bin Wang, Shihua Zhou, Ben Cao, Pan Zheng, and Qiang Zhang. Dvoug enables robust dna sequence assembly and reconstruction with a dynamic, variable-order graph. *Cell Reports Methods*, 5(12):101243, 2025. doi:10.1016/j.crmeth.2025.101243.

21 Eugene W Myers. Toward simplifying and accurately formulating fragment assembly. *Journal of Computational Biology*, 2(2):275–290, 1995. doi:10.1089/cmb.1995.2.275.

22 Gonzalo Navarro and Veli Mäkinen. Compressed full-text indexes. *ACM Comput. Surv.*, 39(1):2, 2007. doi:10.1145/1216370.1216372.

23 Sergey Nurk, Brian P. Walenz, Arang Rhie, Mitchell R. Vollger, Glennis A. Logsdon, Robert Grothe, Karen H. Miga, Evan E. Eichler, Adam M. Phillippy, and Sergey Koren. Hicanu: accurate assembly of segmental duplications, satellites, and allelic variants from high-fidelity long reads. *Genome Research*, 30(9):1291–1305, 2020. URL: https://genome.cshlp.org/content/30/9/1291.

24 Jannik Olbrich. Fast and Memory-Efficient BWT Construction of Repetitive Texts Using Lyndon Grammars. In Anne Benoit, Haim Kaplan, Sebastian Wild, and Grzegorz Herman, editors, *Proc. 33rd Annual European Symposium on Algorithms (ESA)*, volume 351 of *LIPIcs*, pages 60:1–60:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.ESA.2025.60.

25 Brian D. Ondov, Todd J. Treangen, Páll Melsted, Adam B Mallonee, Nicholas H Bergman, Sergey Koren, and Adam M. Phillippy. Mash: fast genome and metagenome distance estimation using minhash. *Genome biology*, 17(1):132, 2016. URL: https://link.springer.com/article/10.1186/s13059-016-0997-x.

26 Yu Peng, Henry C.M. Leung, Siu-Ming Yiu, and Francis Y.L. Chin. IDBA–a practical iterative de Bruijn graph de novo assembler. In Bonnie Berger, editor, *Proc. 14th Annual International Conference on Research in Computational Molecular Biology (RECOMB)*, volume 6044 of *Lecture Notes in Computer Science*, pages 426–440. Springer, 2010. doi:10.1007/978-3-642-12683-3_28.

27 Pavel A. Pevzner, Haixu Tang, and Michael S. Waterman. An eulerian path approach to DNA fragment assembly. *Proceedings of the National Academy of Sciences*, 98(17):9748–9753, 2001. doi:10.1073/pnas.171285098.

28 Martin Raab and Angelika Steger. "Balls into bins" — a simple and tight analysis. In Michael Luby, José D. P. Rolim, and Maria J. Serna, editors, *Proc. 2nd International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)*, volume 1518 of *Lecture Notes in Computer Science*, pages 159–170, 1998. doi:10.1007/3-540-49543-6_13.

29 Mikko Rautiainen and Tobias Marschall. MBG: Minimizer-based sparse de Bruijn graph construction. *Bioinformatics*, 37(16):2476–2478, 2021. doi:10.1093/bioinformatics/btab004.

30 Alexandru I. Tomescu and Paul Medvedev. Safe and complete contig assembly through omnitigs. *Journal of Computational Biology*, 24(6):590–602, 2017. doi:10.1089/cmb.2016.0141.

31 Aaron M. Wenger, Paul Peluso, William J. Rowell, Chang, et al. Accurate circular consensus long-read sequencing improves variant detection and assembly of a human genome. *Nature Biotechnology*, 37:1155–1162, 2019. doi:10.1038/s41587-019-0217-9.

32 Daniel R. Zerbino and Ewan Birney. Velvet: Algorithms for de novo short read assembly using de bruijn graphs. *Genome Research*, 18(5):821–829, 2008. doi:10.1101/gr.074492.107.


# A Appendix

**Table A1** General parameter set for the experiments. For `Ryu`, we divide the general case of choosing the pair $(\ell, h)$ manually and the parameters chosen for calculating the theoretical best. In both cases, a parameter `-r` has been set to discard $(\ell, h)$-tigs smaller than 1000 bases to speed up calculations. The number of threads for `Ryu` is set to 4 since a number of threads larger than the size of the considered alphabet will not yield any improvements.

| Tool | Parameters | Values | Notes |
|---|---|---|---|
| `Ryu` | $\ell$ | 9 9 13 13 17 17 20 24 | Threads = 4<br>`-r = 1000` |
| | $h$ | 16 17 22 25 28 31 26 32 | |
| | Confidence Rate | 0.95 | |
| | Error rate | 0.001 | |
| | Estimated Length | depends on the dataset | |
| `Bcalm2` | `-kmer-size` | 16 20 32 40 64 96 128 150 160 192 200 224 256 | Threads = 24 |
| | `-abundance-min` | 7 9 11 13 15 | |
| `Flye` | | No specified parameters | Threads = 24 |
| `Hifiasm` | | No specified parameters | Threads = 24 |

**Table A2** Comparison between all the instances of `Ryu` with `Hifiasm`, `Flye` and the best two instances of `Bcalm2`. For `Ryu`, ★, ◆, and ▲ indicate the best, fastest, and theoretical optimum assemblies, respectively. The fastest (◆) assembly coincides with the recommended parameters on all datasets.

| Dataset | Tool | Parameter | N50 (Kbp) | Misassemblies | Largest Contig (Kbp) | Total Contigs |
|---|---|---|---|---|---|---|
| ECOLI | Bcalm2 | 128,13 | 1.20 | 0 | 7.07 | 3,301 |
| | | 224,15 | 2.19 | 0 | 15.12 | 2,659 |
| | Flye | | 1,182.54 | 1 | 2,087.87 | 7 |
| | Hifiasm | | 4,665.56 | 1 | 4,665.56 | 1 |
| | Ryu | 9,16 | 13.29 | 0 | 25.67 | 11 |
| | | 9,17 | 13.29 | 0 | 25.67 | 10 |
| | | 13,22 | 4,654.42 | 1 | 4,654.42 | 8 |
| | | 13,25 | 24.55 | 0 | 24.55 | 7 |
| | | 17,28 | 4,640.70 | 1 | 4,640.70 | 6 |
| | | 17,31 ★ | 4,665.84 | 1 | 4,665.84 | 4 |
| | | 20,26 | 4,640.70 | 1 | 4,640.70 | 6 |
| | | 20,28 ▲ | 4,640.70 | 1 | 4,640.70 | 4 |
| | | 24,32 | 4,640.70 | 1 | 4,640.70 | 5 |
| | | 100,150 ◆ | 4,640.70 | 3 | 4,640.70 | 4 |
| YEAST | Bcalm2 | 224,13 | 1.43 | 0 | 9.66 | 9,753 |
| | | 256,15 | 0.89 | 1 | 15.85 | 22,545 |
| | Flye | | 759.17 | 8 | 1,471.70 | 23 |
| | Hifiasm | | 138.67 | 488 | 1,357.40 | 890 |
| | Ryu | 9,16 | 47.43 | 87 | 505.03 | 3,790 |
| | | 9,17 | 58.66 | 85 | 590.06 | 3,673 |
| | | 13,22 | 70.44 | 70 | 391.44 | 2,508 |
| | | 13,25 ★ | 104.14 | 64 | 626.48 | 2,322 |
| | | 17,28 | 71.48 | 66 | 381.41 | 1,825 |
| | | 17,31 | 100.67 | 60 | 511.24 | 1,675 |
| | | 20,26 | 60.30 | 77 | 322.16 | 1,848 |
| | | 23,33 ▲ | 65.33 | 73 | 322.15 | 1,476 |
| | | 24,32 | 64.83 | 74 | 322.15 | 1,484 |
| | | 225,338 ◆ | 23.31 | 21 | 322.04 | 2,565 |
| HUMAN | Bcalm2 | 32,7 | 1.73 | 0 | 25.15 | 827,231 |
| | | 40,7 | 1.93 | 0 | 28.41 | 846,616 |
| | Flye | | 5,953.40 | 238 | 26,830.28 | 2,085 |
| | Hifiasm | | 3,967.22 | 241 | 33,485.86 | 1,453 |
| | Ryu | 3,5 ★◆ | 253.98 | 55 | 1,900.07 | 62,016 |
| | | 5,9 | 84.33 | 80 | 674.80 | 141,087 |
| | | 9,16 | 18.72 | 11 | 169.04 | 221,144 |
| | | 9,17 | 19.03 | 74 | 169.04 | 212,741 |
| | | 13,22 | 6.04 | 0 | 75.77 | 103,080 |
| | | 13,25 | 6.46 | 33 | 78.88 | 88,931 |
| | | 15,20 ▲ | 3.92 | 0 | 53.08 | 61,922 |
| | | 17,28 | 3.12 | 2 | 53.08 | 34,444 |
| | | 17,31 | 3.32 | 5 | 62.87 | 31,029 |
| | | 20,26 | 2.53 | 0 | 47.72 | 25,504 |
| | | 24,32 | 2.22 | 0 | 35.24 | 20,426 |

# 10-Minimizers: A Promising Class of Constant-Space Minimizers

**Arseny Shur**
Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel

**Ido Tziony**
Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel

**Yaron Orenstein**
Department of Computer Science and The Mina and Everard Goodman Faculty of Life Sciences, Bar-Ilan University, Ramat Gan, Israel

## Abstract

Minimizers are sampling schemes ubiquitous in high-throughput sequencing analysis. Given an alphabet of size $\sigma$, a minimizer is defined by two positive integers $k$ and $w$, and a linear order $\rho$ on $k$-mers. A sequence is processed by a sliding window algorithm that chooses, in each window of length $w + k - 1$, its minimal $k$-mer with respect to $\rho$. A key characteristic of a minimizer is its density, defined as the expected frequency of chosen $k$-mers among all $k$-mers in a random infinite $\sigma$-ary sequence. Minimizers of lower density are preferred as they produce smaller samples, which lead to reduced runtime and memory usage in downstream applications. Recently developed methods generate minimizers with optimal and near-optimal densities, but these methods require explicit storage of $k$-mer ranks in $\Omega(2^k)$ space. Methods generating constant-space minimizers with low densities also exist, and some are asymptotically optimal as $k \to \infty$. However, in the non-asymptotic regime, no known class of minimizers had been proven to guarantee, on expectation, a lower density compared to a random minimizer.

In this paper, we introduce a class of 10-minimizers, which has promising properties. First, we prove that for every $k > 1$ and every $w \geq k - 2$, a random 10-minimizer has, on expectation, lower density than a random minimizer, under essentially the same simplifying assumption. This is the first provable guarantee for a class of minimizers in the non-asymptotic regime. Second, we present spacers, which are particular 10-minimizers combining three desirable properties: constant space usage, low density, and short $k$-mer key-retrieval time. In terms of density, spacers are competitive to the best known constant-space minimizers; in certain $(k, w)$ regimes they achieve the lowest density among all known minimizers. Notably, we are the first to benchmark minimizers by the time spent for $k$-mer key retrieval, which is the most fundamental operation in many minimizers-based methods. We propose this benchmark as a standard objective for evaluating new minimizer schemes. Our empirical results show that spacers retrieve $k$-mer keys in competitive time – a few seconds per genome-size sequence – for all practical values of $k$ and $w$. We expect 10-minimizers to improve minimizers-based methods, especially those using large window sizes.

**2012 ACM Subject Classification** Theory of computation → Design and analysis of algorithms; Applied computing → Molecular sequence analysis

**Keywords and phrases** Minimizer, constant-space minimizer, local selection scheme, $k$-mer key retrieval, high-throughput sequencing

**Digital Object Identifier** 10.4230/LIPIcs.WABI.2026.5

**Supplementary Material** *Software (Source code and tables)*: `https://github.com/OrensteinLab/10-minimizers`, archived at `swh:1:dir:1a53d6e21b4b8a7d491152c78d360c1a688d240b`

**Funding** *Yaron Orenstein*: This study was supported by Israel Science Foundation grant no. 358/21.

**Acknowledgements** Ido Tziony acknowledges the VATAT fellowship for excellent PhD students in Data Science, the Bar-Ilan University presidential fellowship, and the cloud-computing credit support of the Israel Data Science and AI Initiative.



© Arseny Shur, Ido Tziony, and Yaron Orenstein;
licensed under Creative Commons License CC-BY 4.0





**Author contribution** A.S. performed the theoretical analysis and led the project, A.S. and I.T. developed, implemented, and benchmarked the algorithms, all authors analyzed the results and wrote the manuscript, and Y.O. supervised I.T. and acquired the funding.

## 1 Introduction

Sampling short substrings, termed $k$-mers, in long DNA sequences is a critical step in solving many bioinformatics tasks. Typically, a "window guarantee" is required: each window of $w$ consecutive $k$-mers (i.e., a window of length $w+k-1$) in the input sequence should be represented by at least one $k$-mer in the sample. Minimizers, introduced in [20,22], are simple sampling schemes with the window guarantee: a linear order on $k$-mers is fixed, and in each window, the minimal (lowest-ranked) $k$-mer w.r.t. this order is selected, with ties broken to the left. Minimizers are the most popular $k$-mer sampling schemes (a comprehensive list of bioinformatics applications using minimizers can be found in [16]), and even more advanced sampling schemes still use minimizers as an intermediate step [2, 10, 21].

Minimizers are most commonly characterized by their density, which is the expected fraction of sampled $k$-mers in an infinite sequence of i.i.d. symbols, as lower density leads to reduced runtime and memory usage of downstream applications. The average density (a.k.a. the expected density of a random minimizer) is close to $\frac{2}{w+1}$ unless $w \gg k$ [5, 22]. Many methods were designed to generate low-density minimizers. Methods like DOCKS [18], PASHA [3], and GreedyMini [6], generate orders explicitly, which highly limits the $k$ values that can be used due to both the exponential, in $k+w$, construction time and $\Theta(\sigma^k)$ space occupied by the order itself. GreedyMini pushes this bottleneck by using binary projections of DNA orders, which require $\Theta(2^k)$ space, but this is still restrictive.

Methods like miniception [23], double-decycling [19], and open-closed syncmer [8], define minimizer orders via functions that assign to each $k$-mer a unique numeric *key* such that the order of $k$-mers is the order of their keys. For example, for a hash-based random minimizer, the key of a $k$-mer is its hash value, while double-decycling minimizers use the "class" of the $k$-mer as the primary key and the hash value as the secondary key, to break ties on primary keys. The minimizers obtained in such a way require only constant space to represent them. The constant-space property enables the application of the minimizers to any value of $k$.

All the methods above generate, for each pair $(k, w)$, a *class* of minimizers, parameterized in some way: by the auxiliary order for miniception and open-closed syncmer; by the secondary key choice for double-decycling; by the sequence of randomized choices during the course of the algorithm for GreedyMini. In all cases, proven density guarantees (like average-case density over the class) would be desirable, but presently do not exist. For miniception [23] and the mod-sampling scheme [10], *asymptotic* density estimates were proven. However, for practical $(k, w)$ values, these estimates are very loose, and thus of little use[1].

As minimizers are intended for sampling, their sampling time is another very important parameter besides density. Namely, the methods can be ranked by the time required to the key retrieval, which is the method-specific part of sampling. For all methods with explicitly stored orders, every key is retrieved by one memory access. However, the key retrieval in constant-space methods varies a lot, and a method with low empirical density still can slow down an application due to a complex key computation. Previously, benchmarking

[1] For example, for $k = 15$ and $w = 10$ (the default parameters used in Minimap [13] and Minimap2 [14]), the measured density of the mod-sampling scheme with the default parameter $t = 4$ is $\approx 0.143$ [6, Fig. 3B], while [10, Thm. 10] gives the estimate $\approx 0.177$ plus an $o$-term, which means the error of 25%.

did not isolate $k$-mer key-retrieval time. Instead, evaluations focused on the total time required to find the minimizer - a metric heavily dependent on the specific implementation of the minimization algorithm (which, for instance, accounts for 80% of the runtime in SimdMinimizers [9]).

In this paper, we present 10-minimizers, a novel class of minimizers with the first provable guarantee of achieving density better than random in the non-asymptotic regime. We prove that random 10-minimizers have the expected density $\approx \frac{2}{w+2}$ compared to $\approx \frac{2}{w+1}$ achieved by random minimizers. We present a particular family of constant-space 10-minimizers, called *spacers*, that achieve densities competitive to those of the known advanced constant-space minimizers. Moreover, for some $(k, w)$ pairs, spacers achieve the lowest density compared to all known (not necessarily constant-space) minimizers. Last, we formally define a performance-evaluation criterion for minimizers: $k$-mer key-retrieval time, and demonstrate that spacers, as well as random 10-minimizers, are highly competitive. In particular, spacers are faster than all constant-space methods that reach, for some $(k, w)$ ranges, lower density. We expect 10-minimizers to improve the runtime and memory usage of many high-throughput sequencing methods by a simple replacement of their current $k$-mer key calculations.

## 2 Preliminaries

In what follows, $\Sigma, \sigma, k$, and $w$ denote, respectively, the alphabet $\{0, \ldots, \sigma-1\}$, its size, the length of the sampled substrings ($k$-mers), and the number of $k$-mers in a window. We write $s[1..n]$ for a length-$n$ string, denote the length of $s$ by $|s|$, and use standard definitions of substring, prefix, and suffix of a string. The empty string is denoted by $\varepsilon$. We write $n$-string ($n$-prefix, $n$-suffix, $n$-window) to indicate length. We use the notation $[i..j]$ for the range of integers from $i$ to $j$, and $s[i..j]$ for the substring of $s$ covering the range $[i..j]$ of positions.

Let $\rho$ be a permutation of $\Sigma^k$, which we view interchangeably as a linear order and as a string of distinct $k$-mers. Formally, we define $\rho$ as a bijection $\rho : [1..\sigma^k] \to \Sigma^k$, binding all $k$-mers to their *$\rho$-ranks*. The *minimizer* $(\rho, w)$ is a map $f : \Sigma^{w+k-1} \to [1..w]$ assigning to each $(w+k-1)$-window the starting position of its minimum-$\rho$-rank $k$-mer, with ties broken to the left. This map acts on strings (which we interchangeably call sequences) over $\Sigma$, selecting one position in each window so that in a window $v = S[i..i+w+k-2]$ the position $i + f(v) - 1$ in $S$ is selected.

Minimizers are evaluated by the density of selected positions. Let $f(S)$ denote the set of positions selected in a string $S$ by a minimizer $f$. The *density of $f$* is the limit $d_f = \lim_{n\to\infty} \frac{1}{\sigma^n} \sum_{S\in\Sigma^n} \frac{|f(S)|}{n}$ (also termed the *expected* density, to distinguish from the *particular* density over a specific sequence). Studying minimizer densities over a range of $w$ values, it is convenient to "normalize" density to the *density factor* $D_f = (w+1)d_f$.

An *arrangement* $\pi$ of $\Sigma^k$ with the *domain* $\varnothing \neq U \subseteq \Sigma^k$ is a permutation of $U$. In particular, arrangements with the domain $\Sigma^k$ are exactly (linear) orders. We view $\pi$ as a bijection $\pi : [1..|U|] \to U$, and apply the notion of $\pi$-rank to arbitrary arrangement $\pi$. We also consider $\pi$ as an ordered list of length $|\pi| = |U|$. We write $U_\pi$ for the domain of $\pi$, and denote by $\pi_1 \cdot \pi_2$ the standard string concatenation of arrangements $\pi_1$ and $\pi_2$ with disjoint domains, i.e., $U_{\pi_1} \cap U_{\pi_2} = \varnothing$.

A subset $H \subseteq \Sigma^k$ is a *universal hitting set* (UHS) *for $w$* if every $(w+k-1)$-window contains at least one $k$-mer from $H$. For example, $\{00, 01, 11\}$ is a UHS for $\sigma = k = 2$ and any $w > 1$. We call an arrangement $\pi$ a *UHS order for $w$* if $U_\pi$ is a UHS for $w$. Then, any two orders $\pi \cdot \pi_1$ and $\pi \cdot \pi_2$ define the same minimizer for $w$, because the minimum-rank $k$-mer in every window is in $\pi$; we denote this minimizer by $(\pi, w)$.

For a given minimizer $f = (\rho, w)$, a $(w+k)$-window $v$ (which contains two consecutive $(w+k-1)$-windows and often termed *context*) is *charged* if its minimum-rank $k$-mer is either its prefix or its *unique suffix* (i.e., the $k$-suffix of $v$ having no other occurrence in $v$); otherwise, $v$ is *free*. Every string $S$ has exactly $|f(S)| - 1$ charged $(w+k)$-windows [24, Lemma 6]. Since all possible $n$-strings have, in total, the same number of occurrences of each $(w+k)$-window, the density $d_f$ of a minimizer equals the fraction of charged windows in $\Sigma^{w+k}$ [15, 23].

For an arrangement $\pi$ of $\Sigma^k$, a $(w+k)$-window $v$ is *charged by* $\pi$ *(due to* $\pi[i]$*)* if $\pi[i]$ is the $k$-mer of minimum $\pi$-rank in $v$, and is a prefix (*prefix-charged*) or a unique suffix (*suffix-charged*) of $v$. The number of windows over $\Sigma^{w+k}$ charged by $\pi$ is denoted by $\mathsf{ch}_w(\pi)$. A *live* window (w.r.t. a set $U \subset \Sigma^{w+k}$) is a window with no $k$-mer in $U$. If $U \subset \Sigma^k$ is disjoint from $U_\pi$, then $\mathsf{ch}_w(U, \pi)$ denotes the number of windows that are both live w.r.t. $U$ and charged by $\pi$. A (UHS) order $\rho$ is *initiated by* $U$ if some prefix of $\rho$ has the domain $U$.

Let $(\rho, w)$ be a binary minimizer, $\sigma > 2$, and let $h : \Sigma \to \{0, 1\}$ be a surjective *projection* map, extended to all strings over $\Sigma$ letter-wise. A minimizer $(\rho', w)$ over $\Sigma$ is a *$\sigma$-extension of $(\rho, w)$ with the projection $h$* if for any $k$-mers $u, v \in \Sigma^k$ the condition $\mathsf{rank}_\rho(h(u)) < \mathsf{rank}_\rho(h(v))$ implies $\mathsf{rank}_{\rho'}(u) < \mathsf{rank}_{\rho'}(v)$. In other words, two $k$-mers $u$ and $v$ can be compared w.r.t. $\rho'$ by comparing their projections $h(u)$ and $h(v)$ w.r.t. $\rho$, using $\rho'$ only to break ties.

In a similar way, *$k$-extensions* of minimizers are defined. Let $(\rho, w)$ and $(\rho', w)$ be two minimizers over some alphabet $\Sigma$ such that $\rho$ is an order on $\Sigma^k$, $\rho'$ is an order on $\Sigma^{k'}$, and $k' > k$. If for any $k'$-mers $u, v \in \Sigma^{k'}$ the condition $\mathsf{rank}_\rho(u[1..k]) < \mathsf{rank}_\rho(v[1..k])$ implies $\mathsf{rank}_{\rho'}(u) < \mathsf{rank}_{\rho'}(v)$, then $(\rho', w)$ is a *$k$-extension* of $(\rho, w)$.

Every string $s \in \Sigma^*$ is a base-$\sigma$ representation of an integer, denoted by $\mathsf{num}(s)$. If $|s| = |u|$, then $\mathsf{num}(s) < \mathsf{num}(u)$ if and only if $s$ precedes $u$ lexicographically. We use Iverson notation: $[\mathsf{expression}]$ evaluates to 1/0 depending on $\mathsf{expression}$ being true/false.

# 3 Methods

## 3.1 10-minimizers

We start with the case $\Sigma = \{0, 1\}$. Let $\mathsf{IO}_k = \{10u \mid u \in \{0, 1\}^{k-2}\}$; the elements of $\mathsf{IO}_k$ are *10-$k$-mers* ("one-zero $k$-mers", a property that will make low-ranked $k$-mers well spaced).

We consider the binary UHS orders initiated by $\mathsf{IO}_k$. First we solve the problem common to all such orders: to find an arrangement $\tau$ that hits all windows that are live w.r.t. $\mathsf{IO}_k$ and charges only few of them.

▶ **Lemma 1.** *Suppose that $\sigma = 2$, $w \geq k - 2$, $\tau = (1^{k-1}0, 01^{k-2}0, \ldots, 0^{k-2}10, 0^{k-1}1, 1^k, 0^k)$, and $\pi$ is an arrangement of $\mathsf{IO}_k$. Then $\pi \cdot \tau$ is a UHS order for $w$ and $\mathsf{ch}_w(\mathsf{IO}_k, \tau) = w + k + 3 + [w = k - 2]$.*

**Proof.** Since a window is live w.r.t. $\mathsf{IO}_k$ if and only if it contains no $k$-mers beginning with 10, the set $X$ of such live windows is given by $X = \{0^i 1^{w+2-i} u \mid i \in [0..w+2], u \in \{0, 1\}^{k-2}\}$. For $1 \leq i \leq k-1$, any window with the prefix $\tau[i] = 0^{i-1}1^{k-i}0$ does not belong to $X$: the longest prefix of the form $0^i1^j$ in a window beginning with $\tau[i]$ has length $k-1$, while the windows in $X$ have such a prefix of length at least $w + 2 > k - 1$. Hence, $\pi \cdot \tau$ prefix-charges 0 windows due to $\tau[1], \ldots, \tau[k-1]$. As a unique suffix, $1^{k-1}0$ occurs in all live windows of the form $0^i1^{w-i}1^{k-1}0$, $i \in [0..w]$, so it suffix-charges $w + 1$ windows. Each of the $k$-mers $0^i1^{k-i-1}0$, $i \in [1..k-2]$, is a unique suffix of exactly one window from $X$, namely, $0^{w+i-1}1^{k-i}0$. Since a window from $X$ cannot contain both $\tau[i]$ and $\tau[j]$, where $1 \leq i < j \leq k - 1$, each $\tau[i]$, $i \in [2..k - 1]$, suffix-charges one window.

Suppose that an $n$-window $v$ is live w.r.t. $\pi \cdot \tau[1..k{-}1]$ and contains 10. We can write $v = v_1 10 v_2$ for some $v_1, v_2 \in \Sigma^*$ such that $v_1$ does not contain 10. Then, $|v_1| \le k-3$ (otherwise, $v$ contains some $\tau[i]$) and $|v_2| \le k-3$ (otherwise, $v$ contains a $k$-mer from $\pi$). Therefore, $n \le 2k-4 \le w+k-2$. Hence, the live $(w+k)$-windows w.r.t. $\pi \cdot \tau[1..k{-}1]$ form the set $X' = \{0^i 1^{w+k-i} \mid i \in [0..w{+}k]\}$ (and for $(w+k-1)$-windows, the structure of the set is the same). Every window from $X'$ contains at least one of the $k$-mers $\tau[k] = 0^{k-1}1$, $\tau[k{+}1] = 1^k$, $\tau[k{+}2] = 0^k$, implying that $\pi \cdot \tau$ is a UHS for $w$. It remains to note that $\tau[k] = 0^{k-1}1$ charges the windows $0^{k-1}1^{w+1}$ and $0^{w+k-1}1$, $\tau[k{+}1] = 1^k$ charges $1^{w+k}$ and $0^w 1^k$ (the latter one only if it does not contain $0^{k-1}1$, i.e., for $w = k-2$), and $\tau[k{+}2] = 0^k$ charges $0^{w+k}$. Summing up all windows charged due to $\tau$, we get the amount stated in the lemma. ◀

▶ Remark 2. The only $k$-mers from $\tau$ that can occur more than once in a live window w.r.t. $\mathsf{IO}_k$ are $1^k$ and $0^k$.

▶ **Definition 3** (10-orders and 10-minimizers). *An order on $\{0,1\}^k$ is a* 10-order *("one-zero-order") if it has the prefix $\pi \cdot \tau$, where $\pi$ is an arbitrary arrangement of $\mathsf{IO}_k$ and $\tau$ is the arrangement defined in Lemma 1. An order $\rho$ on $\{0,\dots,\sigma-1\}^k$ is a* 10-order *if it is a $\sigma$-extension of a binary 10-order with a projection $h : \{0,\dots,\sigma-1\} \to \{0,1\}$. If, moreover, $|h^{-1}(0)| = |h^{-1}(1)|$, then $\rho$ is a* balanced *10-order. A minimizer defined by a 10-order is called a* 10-minimizer *("one-zero-minimizer").*

## 3.2 Expected density of random 10-minimizers

▶ **Definition 4** (random minimizers). *A* random minimizer *is a random variable equidistributed over all minimizers with given parameters $\sigma, k$, and $w$. A* random 10-minimizer *is a random variable equidistributed over all 10-minimizers with given parameters $\sigma, k$, and $w$, and with a given projection $h$ in the case $\sigma > 2$.*

The expected density of a random minimizer equals $\sigma^{-w-k} \sum_{v \in \Sigma^{w+k}} P(v)$, where $P(v)$ is the probability of the window $v$ being charged [5]. To estimate this expected density, a simplifying assumption was used [15], yielding an error that is small for wide ranges of parameters. Let us call a window *repetitive* if it contains some $k$-mer at least twice, and *repetition-free* if it is not repetitive. Informally, the assumption says "ignore all repetitive $(w{+}k)$-windows". As such windows are rare unless $w \gg k$, the error introduced by this assumption can be proved small. Since any repetition-free $(w{+}k)$-window $v$ contains $(w{+}1)$ distinct $k$-mers, a random minimizer prefix-charges $v$ with probability $\frac{1}{w+1}$ and suffix-charges $v$ with the same probability. Hence, the simplifying assumption can be formalized directly as follows:

**(A)** the random minimizer with the parameters $(\sigma, k, w)$ prefix-charges any $(w{+}k)$-window with probability $\frac{1}{w+1}$ and suffix-charges it with the same probability $\frac{1}{w+1}$.

Assuming (A), one estimates the expectation of the density of a random $(\sigma, k, w)$-minimizer to $\frac{2}{w+1}$. Note that for a $(w{+}k)$-window with $t$ distinct $k$-mers, the probability of being prefix-charged by a random minimizer is $\frac{1}{t}$, while the probability of being suffix-charged is either $\frac{1}{t}$ or 0, depending on the uniqueness of the $k$-suffix. Thus, for repetitive $(w{+}k)$-windows (A) underestimates the probability of being prefix-charged, while the probability of being suffix-charged can be affected in any direction.

In what follows, we formulate similar assumptions for random 10-minimizers and estimate their expected density based on these assumptions. We restrict ourselves to the case $w \ge k-2$. Therefore, $\pi \cdot \tau$ is a UHS order for any arrangement $\pi$ of $\mathsf{IO}_k$.

### 3.2.1 Case $\sigma = 2$

We introduce the simplifying assumption for random binary 10-minimizers, aiming to ignore only a fraction of all repetitive windows. Namely, a $(w{+}k)$-window $v$ is *essentially repetitive* if there exists a $k$-mer $u$ that occurs in $v$ at least twice and is minimal in $v$ in at least one 10-order but not in all 10-orders. Informally, the assumption says "ignore all essentially repetitive $(w{+}k)$-windows".

The status (charged or free) of any window without 10-$k$-mers is the same for any 10-minimizer, and the number of charged windows of this type is computed in Lemma 1. Hence, essentially repetitive windows are those containing some 10-$k$-mer at least twice. If a $(w{+}k)$-window $v$ contains $m$ 10-$k$-mers, each occurring in $v$ exactly once, then $v$ is prefix-charged (respectively, suffix-charged) with probability $\frac{1}{m}$ if its $k$-prefix (respectively, $k$-suffix) is a 10-$k$-mer. Accordingly, we assign the same probabilities to all $(w{+}k)$-windows with $m$ occurrences of 10-$k$-mers, formalizing our assumption as follows:

**(A$^\dagger$)** the random 10-minimizer with the parameters $(2, k, w)$, where $w \geq k{-}2$, prefix-charges (respectively, suffix-charges) any $(w{+}k)$-window $v$ with $m \geq 1$ occurrences of 10-$k$-mers with probability $\frac{1}{m}$ if $v$ has a 10-$k$-mer as a prefix (respectively, suffix).

Using (A$^\dagger$), we get the following estimate, very close to $\frac{2}{w+2}$, for the expected density $\mathcal{R}(2, k, w)$ of a random binary 10-minimizer.

▶ **Theorem 5.** *Under the assumption* (A$^\dagger$)*, the expected density $\mathcal{R}(2, k, w)$ of a random 10-minimizer with the parameters $(2, k, w)$, where $w \geq k{-}2$, approximates to*

$$\mathcal{R}^*(2, k, w) = \frac{2}{w+2} - \frac{1}{2^w(w+2)} + \frac{k+w+3+[w=k-2]}{2^{k+w}}. \tag{1}$$

**Proof.** Let $k$ and $w$ be fixed. For $m \geq 1$, we denote by $\mathsf{IOP}(m)$ (resp., $\mathsf{IOS}(m)$) the set of $(w{+}k)$-windows with $m$ occurrences of 10-$k$-mers, one of which is a prefix (resp., suffix) of the window. The function that deletes the first two letters of the $k$-suffix of a window and appends them to the beginning of the window is an obvious bijection from $\mathsf{IOS}(m)$ to $\mathsf{IOP}(m)$ for every $m \geq 1$. Therefore, $|\mathsf{IOP}(m)| = |\mathsf{IOS}(m)|$. Assuming (A$^\dagger$) and taking Lemma 1 into account, we obtain

$$\mathcal{R}^*(2, k, w) = 2^{-(k+w)}\Big(2 \cdot \sum\nolimits_m \frac{|\mathsf{IOP}(m)|}{m} + k + w + 3 + [w = k-2]\Big). \tag{2}$$

To compute $|\mathsf{IOP}(m)|$, we note that a $(w+k)$-window $v$ belongs to $\mathsf{IOP}(m)$ if and only if $v$ begins with 10 and contains $m$ occurrences of 10 in the prefix $v[1..w{+}2]$. Equivalently,

$$v = 10^{i_1}1^{j_1}0^{i_2}1^{j_2}\cdots 0^{i_m}1^{j_m}x, \text{ where } i_1, \ldots, i_m, j_1, \ldots, j_{m-1} > 0,\ j_m \geq 0,\ |x| = k-2.$$

Respectively, the number of options for $v$ is $2^{k-2}$ times the number of positive integer solutions to the inequality $i_1 + j_1 + \cdots + j_{m-1} + i_m \leq w+1$. This number of solutions is the number of possibilities to choose distinct values for the $(2m-1)$ partial sums

$$i_1,\ i_1 + j_1,\ i_1 + j_1 + i_2,\ \ldots,\ i_1 + j_1 + \cdots + j_{m-1} + i_m,$$

which is $\binom{w+1}{2m-1}$. Hence, $|\mathsf{IOP}(m)| = 2^{k-2}\binom{w+1}{2m-1}$. To obtain (1), we prove a technical claim.

▷ Claim 6. $\sum_{t=1,\, t \text{ odd}}^{n} \frac{2}{t+1}\binom{n}{t} = \frac{2^{n+1}-2}{n+1}$.

Proof. We first observe that (i) $\frac{1}{t+1} = \int_0^1 x^t dx$. Next, $\sum_{t=1}^{n} \binom{n}{t} x^t = (1+x)^n$, $\sum_{t=1}^{n} \binom{n}{t}(-x)^t = (1-x)^n$, and thus we obtain (ii) $2 \cdot \sum_{t=1,\, t \text{ odd}}^{n} \binom{n}{t} x^t = (1+x)^n - (1-x)^n$. As a result, we have

$$\sum_{\substack{t=1 \\ t \text{ odd}}}^{n} \frac{2}{t+1}\binom{n}{t} =_{\text{(i)}} \sum_{\substack{t=1 \\ t \text{ odd}}}^{n} 2\binom{n}{t}\int_0^1 x^t dx = \int_0^1 2 \cdot \sum_{\substack{t=1 \\ t \text{ odd}}}^{n} \binom{n}{t} x^t dx$$
$$=_{\text{(ii)}} \int_0^1 ((1+x)^n - (1-x)^n)dx = \frac{2^{n+1}-1}{n+1} - \frac{1}{n+1} = \frac{2^{n+1}-2}{n+1}.$$

The claim is proved. ◁

To finish the proof, we substitute the formula for $|\mathsf{IOP}(m)|$ into (2), using Claim 6 with $n = w+1$, $t = 2m-1$ to compute the sum (note that $\frac{2}{t+1} = \frac{1}{m}$) and finally obtain (1). ◀

### 3.2.2 Case $\sigma \geq 3$

Let a 10-order $\rho$ on $\Sigma^k$ be the $\sigma$-extension of a binary 10-order $\rho^*$ by a projection $h$, and let $v \in \Sigma^{k+w}$. If the binary window $h(v)$ has a unique minimum $k$-mer in the order $\rho^*$, then $v$ is charged by $\rho$ if and only if $h(v)$ is charged by $\rho^*$. Hence, the random 10-order with the parameters $(\sigma, k, w, h)$ charges $v$ with the same probability as the random 10-order with the parameters $(2, k, w)$ charges $h(v)$, except for two special cases of repetitive windows. If $h(v)$ contains 10-$k$-mers, the special case arises if $h(v)$ contains a repeated 10-$k$-mer; this case is exactly the one covered by the assumption ($\mathrm{A}^\dagger$). If $h(v)$ contains no 10-$k$-mers, its minimum-rank $k$-mer is from $\tau$ (Lemma 1), and hence is the same for all binary 10-orders. By Remark 2, $0^k$ and $1^k$ are the only $k$-mers in $\tau$ that can repeat in $h(v)$, and one of these $k$-mers has the minimum rank in $h(v)$ only if $h(v) = 0^{w+k}$ or $h(v) = 0^i 1^{w+k-i}$, where $i \in [0..k-2]$. This is the second special case where $v$ and $h(v)$ have different probabilities of being charged by the random 10-orders, but it affects a negligibly small set of windows ($k$ distinct projections out of $2^{w+k}$ possible).

Thus, to be consistent with ($\mathrm{A}^\dagger$), we state our assumption for the case $\sigma \geq 3$ as follows:

**($\mathrm{A}^\ddagger$)** the random 10-minimizer with the parameters $(\sigma, k, w, h)$, where $\sigma \geq 3$ and $w \geq k-2$, charges any $(w+k)$-window $v$ with the same probability as the random 10-minimizer with the parameters $(2, k, w)$ charges the window $h(v)$ under the assumption ($\mathrm{A}^\dagger$).

Using ($\mathrm{A}^\ddagger$), we extend Theorem 5 to *balanced* random 10-minimizers.

▶ **Theorem 7.** *Under the assumption* ($\mathrm{A}^\ddagger$)*, the expected density* $\mathcal{R}(\sigma, k, w, h)$ *of a balanced random 10-minimizer with the parameters* $(\sigma, k, w, h)$*, where* $\sigma \geq 3$ *and* $w \geq k-2$*, approximates to* $\mathcal{R}^*(\sigma, k, w, h) = \mathcal{R}^*(2, k, w)$*.*

**Proof.** For $v \in \Sigma^{w+k}$, let $P(v)$ be the probability of $v$ being charged by the random 10-minimizer with the parameters $(\sigma, k, w, h)$, assigned according to ($\mathrm{A}^\ddagger$). Similarly, $P'(z)$ denotes the probability that $z \in \{0,1\}^{w+k}$ is charged by the random 10-minimizer with the parameters $(2, k, w)$, assigned according to ($\mathrm{A}^\dagger$). Then, $\mathcal{R}^*(\sigma, k, w, h) = \sigma^{-k-w} \sum_{v \in \Sigma^{w+k}} P(v)$ and $\mathcal{R}^*(2, k, w) = 2^{-k-w} \sum_{z \in \{0,1\}^{w+k}} P'(z)$. Since $|h^{-1}(0)| = |h^{-1}(1)| = \frac{\sigma}{2}$, every window $z \in \{0,1\}^{w+k}$ satisfies $|h^{-1}(z)| = \left(\frac{\sigma}{2}\right)^{w+k}$. By ($\mathrm{A}^\ddagger$), all windows with the projection $z$ have the same probability of being charged, equal to $P'(z)$. Then, $\sum_{v \in \Sigma^{w+k}} P(v) = \left(\frac{\sigma}{2}\right)^{w+k} \sum_{z \in \{0,1\}^{w+k}} P'(z)$, implying $\mathcal{R}^*(\sigma, k, w, h) = \mathcal{R}^*(2, k, w)$. ◀

▶ Remark 8. The error of the approximations computed in Theorems 5, 7 can be roughly bounded by the fraction of $(w+k)$-windows with repeated $k$-mers among all $(w+k)$-windows. To evaluate the actual error, we computed the exact expected density in the case where $\sigma = 2$ and $k \in \{7, 12\}$; see Section 4.1.

## 3.3 Low-density 10-minimizers

### 3.3.1 Existing 10-minimizers

In [1], the *ABB order* was proposed in the context of studying codes. This order compares any two $k$-mers over $\Sigma$ using the alphabetic order for their first letters and the rule $0 > 1 = 2 = \cdots = \sigma - 1$ for subsequent letters. Neither [1] nor the later paper [4] (which applied this order as part of a minimizer) specified the tie-break used to make ABB a linear order. In [7, Ch. 8], lexicographic tie-breaking was assumed and the resulting order, termed ABB+, was considered from the density point of view.

For every $k > 1$, the ABB+ order is an "almost" 10-order: it is obtained by $\sigma$-extension of a binary order $\rho$ with the projection $h(0) = 1, h(a) = 0$ for $a = 1, \ldots, \sigma-1$, and $\rho$ is initiated by the set $\mathsf{IO}_k$. The sorting used for the rest of $k$-mers is inefficient compared to the arrangement $\tau$ employed in 10-orders (Lemma 1), but this makes a difference only for small $w$ (for large $w$, the windows with no 10-$k$-mers in their projections are extremely rare). Since ABB+ in most cases has smaller density than the expected density of a random 10-minimizer, it serves as an important baseline for our minimizers.

### 3.3.2 Spacers

For a $k$-mer $u \in \{0,1\}^k$, its *tail* $\mathsf{tail}(u)$ is the longest proper suffix of $u$ that is a prefix of a 10-$k$-mer. For example, $\mathsf{tail}(1001010) = 1010$, $\mathsf{tail}(1001111) = 1$, $\mathsf{tail}(1000000) = \varepsilon$. To obtain binary 10-minimizers of lower density than expected by Theorem 5, we assign lower ranks to the 10-$k$-mers that ensure larger distance to the next 10-$k$-mer in the input sequence. In other words, we prioritize 10-$k$-mers with short tails.

▶ **Definition 9** (binary spacer)**.** Binary spacer $(\rho, w)$ *is a binary 10-minimizer such that $\rho$ ranks all 10-$k$-mers by assigning to each 10-$k$-mer $u$ the triple* $(|\mathsf{tail}(u)|, -\mathsf{num}(\mathsf{tail}(u)), \mathsf{num}(u))$ *of numeric keys, and comparing these triples lexicographically.*

▶ Remark 10.

**(1)** The primary key $|\mathsf{tail}(u)|$ distinguishes binary spacers from binary ABB+ orders. For example, a binary spacer assigns the $k$-mer 101110 lower rank compared to 100100, while binary ABB+ (assuming $1 < 0$) ranks these $k$-mers in reverse order.

**(2)** Using the inverse lexicographic order on tails as the secondary key showed the best empirical density results among the "simplest" tie-breakers.

**(3)** The choice of tertiary key has a near-zero impact on the density (e.g., for $k = 12$ and $10 \le w \le 36$ choosing the inverse lexicographic order for the tertiary key instead of the lexicographic order changes the density by $< 0.04\%$ in all cases); the only purpose of this key is to create a linear order.

▶ **Proposition 11.**

**(1)** *Any binary spacer $(\rho, w)$ is a constant-space minimizer.*

**(2)** *Assuming that a $k$-mer fits into $O(1)$ machine words, any two $k$-mers can be compared w.r.t. the order $\rho$ in $O(1)$ time.*

**Proof.**

**(1)** Definition 9 provides the required $O(1)$-size description for all binary spacers.

**(2)** It suffices to show that $|\mathsf{tail}(u)|$ can be found from the 10-$k$-mer $u$ in $O(1)$ operations. We compute the "indicator" $\mathsf{ind}(u) = \mathsf{num}(u)\,\&\,{\sim}(\mathsf{num}(u) \ll 1)$ (here $\&$, $\sim$, and $\ll$ are the bitwise and, bitwise not, and left shift, respectively). The positions of 1's in $\mathsf{ind}(u)$ are exactly the positions of 10's in $u$. Then the second, from the left, occurrence of 1 in $\mathsf{ind}(u)$ is at the position of $\mathsf{tail}(u)$ in $u$ by the definition of tail. Two more operations are needed to locate this position: a xor operation with the constant $2^{k-1}$ removes the leftmost 1, and the number of leading zeroes in the fixed-length representation of the obtained number is computed by the instruction lzcnt, which is standard for modern CPUs. ◀

Binary spacers have significantly lower density than binary ABB+. For example, for $w = 24$ and $k \in [8..26]$, binary spacers have the density factors $\approx 1.74$, compared to $\approx 1.86$ for ABB+ and $\approx 1.92$ for the random 10-minimizer according to Theorem 5 (Supplementary Table A2).

### 3.3.3 DNA spacers

As DNA minimizers are our primary interest, we consider $\sigma$-extensions of binary spacers. Such a $\sigma$-extension is defined by adding a tie-breaker for $k$-mers with the same projection, to create a linear order. This tie-breaker makes the tertiary key of the binary spacer redundant, as the only purpose of the latter is to create a linear order; see Remark 10(3). Taking this observation into account, we define spacers over larger alphabets as follows.

▶ **Definition 12** (spacer). *Let an alphabet $\Sigma$ of size $\sigma > 2$ and a projection map $h : \Sigma \to \{0, 1\}$ be given. A $k$-mer $u \in \Sigma^k$ is* 10-projected *if $h(u)$ is a 10-$k$-mer.* Spacer $(\rho, w)$ *over $\Sigma$ with the projection $h$ is a 10-minimizer such that $\rho$ ranks 10-projected $k$-mers by assigning to each such $k$-mer $u$ the triple $(|\mathsf{tail}(h(u))|, -\mathsf{num}(\mathsf{tail}(h(u))), \mathsf{num}(u))$ of numeric keys, and comparing these triples lexicographically.*

If a spacer is balanced (i.e., if $h$ is balanced), then it inherits the density of the corresponding binary spacer almost exactly (for the details on the density guarantees for balanced $\sigma$-extensions see [6, Sect. 3.3]). On the other hand, $\sigma$-ary ABB+ minimizers, being *unbalanced* $\sigma$-extensions of binary minimizers, show much lower densities than in the binary case (density factor $\approx 1.72$–$1.73$ for the above-mentioned range $w = 24$, $k \in [8..26]$, and the DNA alphabet). To reach even lower density, we focus on unbalanced spacers. For the 4-ary alphabet, we define the *DNA spacer* by the projection $h(0) = h(1) = h(2) = 0$, $h(3) = 1$.

▶ Remark 13. Given a sequence $x$ of $n$ bits, the subsequence of bits in even positions can be extracted from $x$ in $O(\log n)$ bitwise operations by a standard FFT-like algorithm. As a DNA $k$-mer $u$ is a sequence of $2k$ bits, its projection $h(u)$ can be found in $O(\log k)$ operations by extracting the bits in even positions from the sequence $(u\,\&\,(u \ll 1)) \gg 1$.

Proposition 11, Remark 13, and Definition 12 together imply

▶ **Proposition 14.**

**(1)** *Any DNA spacer $(\rho, w)$ is a constant-space minimizer.*

**(2)** *Assuming that a $k$-mer fits into $O(1)$ machine words, any two $k$-mers can be compared w.r.t. the order $\rho$ in $O(\log k)$ time.*

As we demonstrate in Section 4.2, DNA spacers reach lower density than ABB+ DNA minimizers for all $(w, k)$ regimes.

## 3.4 $k$-mer key retrieval

### 3.4.1 Key retrieval problem

Fast sequence sampling with a minimizer $(\rho, w)$ requires efficient solutions to two problems. First, the input sequence $S$ should be converted into a sequence of fast-sortable keys, preferably numeric, representing all $k$-mers in $S$. Second, the minimum key in every set of $w$ consecutive keys should be computed; this key represents the minimum $k$-mer w.r.t. $\rho$ in the corresponding $(w{+}k{-}1)$-window. The second problem is independent of $\rho$, so we do not consider it here. Meanwhile, the solution to the first problem is $\rho$-specific unless $\rho$ is explicitly stored. To be practically useful, a constant-space minimizer should be endowed with a fast key-retrieval algorithm. Let us formally define the problem.

▶ **Definition 15** ($k$-mer key-retrieval problem for DNA minimizers)**.**
**Parameters:** *$\sigma = 4$, $k \geq 2$, $w \geq 2$, an order $\rho$ on $\Sigma^k$.*
**Input:** *an arbitrary sequence $S = S[1..n] \in \Sigma^*$.*
**Output:** *a sequence $K = K[1..n{-}k{+}1]$ of $k$-*mer keys*, satisfying the following property:*
($*$) *for every $(w{+}k{-}1)$-window $v = S[i..i{+}w{+}k{-}2]$ in $S$ and every $j \in [i..i{+}w{-}1]$, the $k$-mer $S[j..j{+}k{-}1]$ has the minimum $\rho$-rank among the $k$-mers in $v$ if and only if $K[j]$ is the minimum key among $K[i], \ldots, K[i{+}w{-}1]$.*

▶ Remark 16. The condition ($*$) ensures that $K$ suffices for a correct sampling of $S$ with the minimizer $(\rho, w)$. At the same time, ($*$) gives a lot of freedom in assigning keys to $k$-mers. In particular, no bijection between $k$-mers and keys is required. For example, if $S[j..j{+}k{-}1]$ provably cannot be a minimum $k$-mer in any window it occurs in, then $K[j]$ can be safely set to be the "infinite" key.

We suggest the key-retrieval time as an important metric for local sampling schemes, in particular, minimizers. For minimizers, we assume standard conditions for all methods:
**(1)** Input sequence $S$ is processed left to right, one symbol at a time
**(2)** Each iteration consists of
- reading the next symbol and updating the $k$-suffix of the input read so far
- computing and reporting the key of the $k$-suffix

Under such conditions, the difference of key-retrieval time between methods reflects the complexity of computing the key, which is exactly the method-specific parameter we suggest to measure. In addition, we get a natural lower bound for all methods: the time needed by the minimizer with **no** key computation, which is exactly the lexicographic minimizer (it reports the $k$-suffix as the key).

The key assignment and key retrieval for DNA spacers are described below, and the results of benchmarking of all competitive methods are given in Section 4.3.

### 3.4.2 Assigning and retrieving $k$-mer keys for DNA spacers

▶ **Definition 17** ($k$-mer keys assignment for DNA spacers)**.** *The key of a $k$-mer $u$ is a pair $(\mathsf{pkey}(u), \mathsf{num}(u))$ of integers, where the* primary key $\mathsf{pkey}(u)$ *is assigned as follows.*
**(1)** *If $u$ is 10-projected, then $\mathsf{pkey}(u)$ represents the pair $(|x|, -\mathsf{num}(x))$, where $x = \mathsf{tail}(h(u))$; we assign $\mathsf{pkey}(u) = |x| \cdot 2^{k-2} + 2^{k-2} - \mathsf{num}(x)$;*
**(2)** *If $u$ is not 10-projected, then $\mathsf{pkey}(u)$ is one of five predefined constants $X_1 < X_2 < X_3 < X_4 < X_\infty$, where $X_1 > \mathsf{pkey}(u)$ for any 10-projected $k$-mer $u$; specifically,*
- *$X_1$ is assigned if $h(u) \in \tau[1..k{-}1]$;*
- *$X_2$ is assigned if $h(u) = \tau[k] = 0^{k-1}1$;*
- *$X_3$ is assigned if $h(u) = \tau[k{+}1] = 1^k$;*

- $X_4$ *is assigned if* $h(u) = \tau[k{+}2] = 0^k$*;*
- $X_\infty$ *is assigned if* $h(u) \notin \tau$*;*

($\star$) *in addition,* $X_\infty$ *is assigned instead of* $X_1$–$X_4$ *if at the moment of assignment* $u$ *is known to be* ***non-minimum*** *in every window it appears in.*

▶ Remark 18.

**(i)** The $k$-mers with $\mathsf{pkey} = X_1$ have projections ending in 10. Hence, any two $k$-mers with $\mathsf{pkey} = X_1$ are separated by at least one 10-projected $k$-mer (which has a smaller $\mathsf{pkey}$ by the choice of $X_1$). Then, two $k$-mers with $\mathsf{pkey} = X_1$ are never compared for the minimum in a window. This property justifies the use of the same $\mathsf{pkey}$ for the $k$-mers with the projections from $\tau[1..k{-}1]$.

**(ii)** If $w \geq k-2$, Lemma 1 ensures that every $(w{+}k{-}1)$-window contains a $k$-mer with the projection in $\mathsf{IO}_k$ or in $\tau$. Hence, tie breaks for the $k$-mers with $\mathsf{pkey} = X_\infty$ never affect the computation of the minimum $k$-mer in a window.

▶ **Proposition 19.** *Any algorithm assigning the $k$-mer keys according to Definition 17 solves the key-retrieval problem for DNA spacers.*

**Proof.** From Definition 12 and Remark 18, it follows that, for any input sequence $S$, the sequence $K$ of keys assigned according to Definition 17 satisfies Definition 15($*$) for $(\rho, w)$ being the DNA spacer. Since ($*$) ensures the correct sampling of $S$, the proposition holds. ◀

Now we describe KRspacer, a fast key-retrieval algorithm for DNA spacers. It obeys conditions (1,2) (Section 3.4.1) so that each iteration follows the scheme

$$\text{read a symbol } a \to \text{update the } k\text{-suffix } u \text{ and its projection } h(u) \text{ using } a \to$$
$$\to \text{compute } \mathsf{pkey} \text{ from } h(u) \to \text{report } \mathsf{pkey}, \mathsf{num}(u)\,.$$

A *trivial* iteration skips the $\mathsf{pkey}$ computation and reports $\mathsf{pkey} = X_\infty$ immediately after the update step. All other iterations are *nontrivial*. KRspacer uses Definition 17($\star$) to make most of the iterations trivial.

**(1)** KRspacer has three *states* and works in *phases*; each phase consists of a nontrivial iteration followed by zero or more trivial iterations and the state update;

**(2)** state10 indicates that the $k$-mer $u$ processed in the nontrivial iteration is 10-projected;
- in the nontrivial iteration, $\mathsf{pkey}$ is computed by Definition 17(1);
- after that, the algorithm branches on $t = |\mathsf{tail}(h(u))|$;
- if $t \geq 2$: perform $k-t-1$ trivial iterations, remain in state10;
- if $t = 0$: switch to state0;
- if $t = 1$: set the counter $i$ to be the number of trailing 1's in $h(u)$, switch to state1;

**(3)** state0 indicates $h(u) = 0^{k-1}h(a)$ for the $k$-mer $u$ processed in the nontrivial iteration;
- after finishing the nontrivial iteration in accordance with Definition 17(2), the algorithm remains in state0 if $h(u) = 0^k$ and switches to state1 with $i = 1$ if $h(u) = 0^{k-1}1$;

**(4)** state1 indicates $h(u) = 0^{k-i-1}1^i h(a)$ for the $k$-mer $u$ processed in the nontrivial iteration;
- if $h(a) = 0$, the algorithm assigns $\mathsf{pkey} = X_1$ by Definition 17(2), performs $k-3$ trivial iterations and switches to state10;
- if $h(a) = 1$ and $i < k-1$, the algorithm assigns $\mathsf{pkey} = X_\infty$ by Definition 17(2), increments $i$ and remains in state1;
- if $h(a) = 1$ and $i = k-1$, the algorithm assigns $\mathsf{pkey} = X_3$ by Definition 17(2) and remains in state1;

**(5)** Initialization: read $S[1..k{-}3]$ and start the first iteration with reading $S[k{-}2]$ in state10; throw the first two computed keys as they do not represent $k$-mers.

▶ **Proposition 20.** KRspacer *solves the key-retrieval problem for DNA spacers.*

**Proof.** By Proposition 19, it suffices to prove that KRspacer assigns keys to all $k$-mers in $S$ according to Definition 17. First we show that all transitions between states are correct. The transitions from state10 follow the definition of tail. If $\mathsf{tail}(h(u))$ begins with 10 (i.e., $t = |\mathsf{tail}(h(u))| \geq 2$), it determines the position of the next 10-projected $k$-mer to be $k-t$ symbols to the right of the currently processed $k$-mer $u$. Hence, it is correct to be in state10 after $k-t-1$ trivial iterations. If $\mathsf{tail}(h(u)) = \varepsilon$, then $h(u) = 10^{k-1}$, so KRspacer correctly switches to state0 for the next phase. Similarly, $\mathsf{tail}(h(u)) = 1$ implies $h(u) = 10^{k-i-1}1^i$, and KRspacer respectively switches to state1 with the correct value of $i$. The transitions from state0 and state1 are checked in the same way.

In nontrivial iterations, the keys are assigned by Definition 17. It remains to check that the assignment of infinite keys in trivial iterations satisfies the condition $(\star)$ of Definition 17. Let $I$ be the set of consecutive $k$-mers that obtained infinite keys during trivial iterations of one phase. Note that the elements of $I$ are not 10-projected. If the phase corresponds to state10, then $|I| = k - t - 1 \leq k - 3 < w$. The $k$-mers in $I$ are preceded and followed in $S$ by 10-projected $k$-mers. By definition, 10-projected $k$-mers have lower ranks than the $k$-mers in $I$. Hence, the assignment of infinite keys respects $(\star)$. Similarly, if the phase corresponds to state1, then $|I| = k - 3 < w$. The $k$-mers in $I$ are preceded by the $k$-mer $u$ with $\mathsf{pkey}(u) = X_1$, and followed by a 10-projected $k$-mer. By Remark 18(i), no $k$-mer in $I$ has $\mathsf{pkey} = X_1$ according to Definition 17(2). Once again, the elements of $I$ are surrounded in $S$ by $k$-mers of lower ranks, and thus the assignment of infinite keys respects $(\star)$. ◀

# 4 Results

## 4.1 Expected density of random 10-minimizers: estimated vs. exact

To evaluate the actual error introduced by assumption $(\mathrm{A}^\dagger)$ and the closely related assumption $(\mathrm{A}^\ddagger)$ to the expected density of a random 10-minimizer, we designed an optimized enumeration algorithm (Supplementary Section A.1) that computes the *exact* expected density of a random 10-minimizer with the parameters $(2, k, w)$ via the sum, over all $(w+k)$-windows containing 10-$k$-mers, of probabilities of being charged by such a random 10-minimizer. Recall that the number of charged windows among those containing no 10-$k$-mers is given by Lemma 1. For $k \in \{7, 12\}$, $w \in [k-2..2k]$, we compared the estimate (1) to the exact expected density (Supplementary Table A1). The error of the estimate is very small: $< 0.11\%$ for $k = 7$ and $< 0.0022\%$ for $k = 12$, which is less than the error of the estimate $\frac{2}{w+1}$ for the density of random minimizers over the same ranges of parameters.

Thus, the estimate $\frac{2}{w+2}$ for the expected density of a random binary 10-minimizer is quite precise not only asymptotically but also for practical $(w, k)$ ranges. This conclusion also extends to random balanced 10-minimizers over arbitrary even-sized alphabets as the simplifying assumption $(\mathrm{A}^\ddagger)$ is based on $(\mathrm{A}^\dagger)$.

## 4.2 Density results for DNA spacers

We compared the density achieved by DNA spacers to densities achieved by other constant-space minimizers, specifically: miniception [23], double-decycling [19], open-closed syncmer [18], and ABB+ [1, 7]. We included two baselines: one is the KGMLT lower bound [11] on the density of all *forward* schemes, including minimizers. The second baseline presents the lowest density achieved by non-constant-space minimizers generated by the state-of-the-art

GreedyMini method [6]. We do not consider mod-sampling-based schemes [10] as they are not minimizers in general[2] and, additionally, they are designed for $k > w$ (even $k \geq w + 4$ in the default setting), while DNA spacers are designed primarily for $w \geq k - 2$.

We measured the density of constant-space minimizers using a random DNA sequence of $2 \times 10^6$ nucleotides. We averaged the results of 5 runs over different sequences, ensuring to use different seeds for the minimizers that rely on randomness, i.e., miniception and open-closed-syncmer. For both miniception and open-closed-syncmer, which rank $s$-mers for some $s < k$, we used $s = 4$. The density measurement for GreedyMini minimizers is detailed in Supplementary Section A.4.

We measured the density in the modes "fixed $k$, varying $w$" for $k \in \{12, 24\}$, $3 \leq w \leq 100$ (Fig. 1 AB), and "fixed $w$, varying $k$" for $w \in \{12, 24\}$ and $3 \leq k \leq 32$ (Fig. 1 CD). For the DNA spacer and $w < k - 2$, we show the density of the $k$-extension (with lexicographic tie-breaking) of the DNA spacer for $(w{+}2, w)$.

For $k = 12$, DNA spacer beats all other constant-space minimizers for $w \geq 23$ and all minimizers, including GreedyMini-generated, for $w \geq 40$. For $k = 24$, double-decycling wins for $w \geq k$, but DNA spacer catches it up as $w$ grows to 100. In the fixed $w$ mode, DNA spacer is still competitive, especially for $k$ small enough. In particular, it beats all constant-space minimizers for $w = 24$, $5 \leq k \leq 12$, and is better than GreedyMini for $k = 5, 6, 7$.

## 4.3 Key-retrieval times benchmarking

The key-retrieval time of a minimizer heavily depends on the hardware, so the absolute values may be insufficiently informative. However, as mentioned in Section 3.4.1, the lexicographical minimizer presents a natural baseline to refer to, as it does not compute $k$-mer keys.

We included in comparison all minimizers from the density benchmarking (DNA spacer, ABB+, miniception, open-closed syncmer, double-decycling, GreedyMini), random 10-minimizer (using a straightforward adaptation of the KRspacer algorithm), and random minimizer. We adapted all sampling algorithms to C++, implementing tricks used in their respective papers and repositories and improving upon them. As several minimizers rely on random orders, we emulated it by first XORing each $k$-mer (or $s$-mer in the case of miniception and open-closed syncmer) with the random seed and then hashing it. We experimented with two hash functions: C++ standard `std::hash`, and `PCG RXS-M-XS` 64-bit mixer [17]; the latter is very fast while satisfying the rigorous TestU01 BigCrush statistical tests [12].

We measured the processing time for a random DNA sequence of $1.5 \times 10^8$ nucleotides, averaging the results over a hundred runs to ensure a robust estimate. All experiments were conducted concurrently and shuffled using 6 cores (out of 8) on an AMD Ryzen 7800X3D system with 32 GB of DDR5 RAM. We measured GreedyMini separately, as it heavily relies on cache space. The results of measurement for $5 \leq k \leq 32$ are shown in Fig. 2. The main conclusions are as follows.

**(1)** Double-decycling, which performs well in terms of density, is very slow. In the key assignment, it relies on precise computations over complex numbers. Apparently, any known method that spends $O(1)$ time per key amplifies the precision error, eventually leading to incorrect key assignment. Otherwise, computing each key requires $\Theta(k)$ operations. In our experiment, all key-retrieval times for double-decycling exceeded $(400k + 200)$ms. Already at $k = 15$, this is four times slower than the second slowest minimizer; hence, we excluded double-decycling from Fig. 2.

[2] In [10, Lemma 13], it is claimed that mod-sampling schemes with a specific choice of the internal parameter are minimizers, but the proof is flawed. See Supplementary Section A.3 for details.

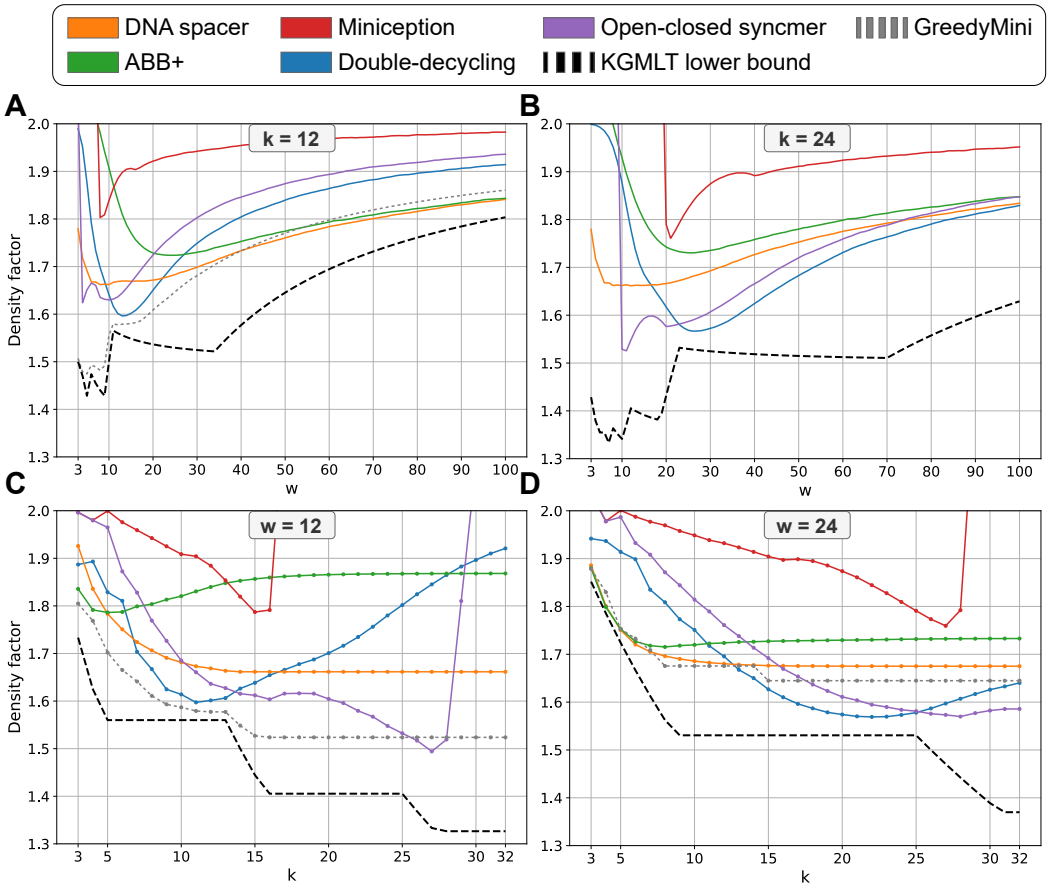



**Figure 1** Performance of various DNA minimizers over certain ranges of $(k, w)$ values. (A-B) Density comparisons over fixed $k \in \{12, 24\}$ and $3 \leq w \leq 100$. GreedyMini is not shown in B since $k = 24$ is beyond its recommended range in terms of both runtime and sampling time. (C-D) Density comparisons over fixed $w \in \{12, 24\}$ and $3 \leq k \leq 32$.

**(2)** The hashing speed has a large effect on key-retrieval times of randomized minimizers. Using `std::hash` instead of the `PCG` mixer increases the time to compute $k$-mer keys (i.e., the distance to the baseline) up to the factor of 4, reached by a random minimizer. The slowing effect is modest for a random 10-minimizer, as it hashes only 10-projected $k$-mers.

**(3)** The fastest constant-space minimizers are the `PCG`-mixer-based random minimizer and ABB+. They are followed by DNA spacer and random 10-minimizer (5–6 seconds per a 2Gbp-long sequence). Miniception and open-closed syncmer are significantly slower[3].

**(4)** For most constant-space minimizers, the key-retrieval time is independent of $k$ as long a $k$-mer fits into a 64-bit machine word. The only exclusions are small $k$ values for DNA-spacer and random 10-minimizer, where the key computation is a bit slower due to a smaller fraction of trivial iterations.

**(5)** Predictably, the time required by non-constant-space minimizers is determined by the cache size and behavior. In our experiment, GreedyMini was the fastest up to $k = 16$ and good up to $k = 21$, but fell out of competition since $k = 23$.

[3] While the pseudocode of the open-closed syncmer suggests that it is slower than miniception, our benchmarking demonstrated variable performance, likely due to the memory layout. See Supplementary Section A.5 for details.

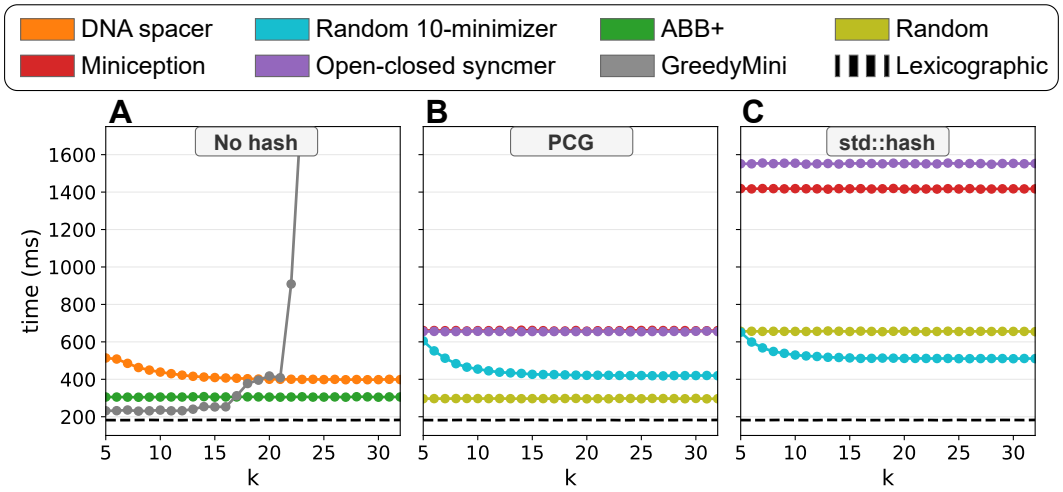



**Figure 2** Key-retrieval times for $k$-mers over a $1.5 \times 10^8$ random DNA sequence, averaged over 100 runs, for $5 \leq k \leq 32$. (A) Minimizers requiring no hash function. (B) Minimizers using the `PCG RXS-M-XS` 64-bit mixer hash function [17]. (C) Minimizers using `std::hash`.

## 5 Discussion

10-minimizers represent a significant advance in $k$-mer sampling schemes for high-throughput sequencing analysis, introducing the first class of constant-space minimizers with provable density guarantees superior to random minimizers in the non-asymptotic regime. By proving that random 10-minimizers achieve an expected density of approximately $\frac{2}{w+2}$ (vs. $\frac{2}{w+1}$ for random minimizers), we established a theoretical foundation for their superiority across practical $k$, $w$ ranges, validated by exact density computations showing approximation errors $< 0.0022\%$ for $k = 12$, $10 \leq w \leq 24$. While the difference between $\frac{2}{w+2}$ and $\frac{2}{w+1}$ seems small, it is not negligible for practical $w$ ranges: e.g., for $k = 12$ the random 10-minimizer achieves lower density than miniception over the whole range of $w$ values. Given their competitive key-retrieval time even for "heavy" hash functions, random 10-minimizers can replace random minimizers in applications that require minimizers randomly chosen from a large set.

DNA spacers, our flagship 10-minimizers, combine three desirable properties: (1) constant space (O(1) description, no explicit order storage); (2) low density, competitive with state-of-the-art schemes, and even achieving the lowest known density for certain $(k, w)$ regimes (e.g., for $40 \leq w \leq 100$ and $k = 12$, or $5 \leq k \leq 7$ and $w = 24$); and (3) fast $k$-mer key retrieval, clearly outperforming constant-space minimizers of lower density. Two key observations that lead to the notion of DNA spacer are: (1) maximization of inter-minimizer distances via short tails prioritization reduces the density; (2) 10-minimizers using unbalanced projections achieve lower density compared to those using balanced projections.

The benchmarking on the new and apparently important metric of key-retrieval time revealed that none of existing methods produces "universally best" minimizers. When both $k$ and $w$ take small/medium values, GreedyMini is a clear winner in both density and key-retrieval time. However, its key-retrieval time for large $k$ is prohibitive, while for large $w$ it loses in terms of density to DNA spacers and ABB+. Double-decycling method is attractive in terms of density but lacks a fast key-retrieval algorithm, and it is unclear whether such an algorithm can exist. Hence, in the "large $k$" and "large $w$" regimes, the main competitors are DNA spacer, ABB+, and open-closed syncmer, with the DNA spacer being arguably the most attractive option for large $w$ values.

There are several open questions and future steps that arise from our study. First, can we push the density of constant-space 10-minimizers further down, while keeping the key-retrieval time on the same level? Second, is it possible to obtain further theoretic density estimates, for example, on the expected density of a random unbalanced 10-minimizer or on the density of a DNA spacer? Third, can we improve current high-throughput sequencing analysis methods and advanced sampling schemes by plugging in the best constant-space minimizer for their $(k, w)$ values?

To conclude, 10-minimizers, particularly spacers, offer a theoretically-grounded, practically-efficient alternative to existing schemes, poised to enhance high-throughput sequencing pipelines through superior density and competitive key-retrieval times.

## References


1 Simon R Blackburn. Non-overlapping codes. *IEEE Transactions on Information Theory*, 61(9):4890–4894, 2015. doi:10.1109/TIT.2015.2456634.

2 Robert Edgar. Syncmers are more sensitive than minimizers for selecting conserved $k$-mers in biological sequences. *PeerJ*, 9:e10805, February 2021. doi:10.7717/peerj.10805.

3 Barış Ekim, Bonnie Berger, and Yaron Orenstein. A randomized parallel algorithm for efficiently finding near-optimal universal hitting sets. In *International Conference on Research in Computational Molecular Biology*, pages 37–53. Springer, 2020.

4 Martin C Frith, Laurent Noé, and Gregory Kucherov. Minimally overlapping words for sequence similarity search. *Bioinformatics*, 36(22-23):5344–5350, 2020. doi:10.1093/BIOINFORMATICS/BTAA1054.

5 Shay Golan and Arseny M. Shur. Expected density of random minimizers. In Rastislav Královic and Vera Kurková, editors, *SOFSEM 2025: Theory and Practice of Computer Science - 50th International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM 2025, Proceedings, Part I*, volume 15538 of *Lecture Notes in Computer Science*, pages 347–360. Springer, 2025. doi:10.1007/978-3-031-82670-2_25.

6 Shay Golan, Ido Tziony, Matan Kraus, Yaron Orenstein, and Arseny Shur. GreedyMini: generating low-density DNA minimizers. *Bioinformatics*, 41(Supplement_1):i275–i284, 2025. doi:10.1093/BIOINFORMATICS/BTAF251.

7 Ragnar Groot Koerkamp. Optimal throughput bioinformatics (doctoral thesis), 2025. doi:10.3929/ethz-c-000783091.

8 Ragnar Groot Koerkamp, Daniel Liu, and Giulio Ermanno Pibiri. The open-closed mod-minimizer algorithm. *Algorithms Mol. Biol.*, 20(1):4, 2025. doi:10.1186/s13015-025-00270-0.

9 Ragnar Groot Koerkamp and Igor Martayan. SimdMinimizers: Computing Random Minimizers, fast. In Petra Mutzel and Nicola Prezza, editors, *23rd International Symposium on Experimental Algorithms (SEA 2025)*, volume 338 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20:1–20:19, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SEA.2025.20.

10 Ragnar Groot Koerkamp and Giulio Ermanno Pibiri. The mod-minimizer: A simple and efficient sampling algorithm for long k-mers. In Solon P. Pissis and Wing-Kin Sung, editors, *24th International Workshop on Algorithms in Bioinformatics, WABI 2024*, volume 312 of *LIPIcs*, pages 11:1–11:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.WABI.2024.11.

11 Bryce Kille, Ragnar Groot Koerkamp, Drake McAdams, Alan Liu, and Todd J Treangen. A near-tight lower bound on the density of forward sampling schemes. *Bioinformatics*, 41(1):btae736, December 2024. doi:10.1093/bioinformatics/btae736.

12 Pierre L'Ecuyer and Richard Simard. Testu01: A c library for empirical testing of random number generators. *ACM Transactions on Mathematical Software*, 33(4):1–40, 2007. doi:10.1145/1268776.1268777.

13 Heng Li. Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. *Bioinformatics*, 32(14):2103–2110, March 2016. doi:10.1093/bioinformatics/btw152.

14 Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18):3094–3100, 2018. doi:10.1093/BIOINFORMATICS/BTY191.

15 Guillaume Marçais, David Pellow, Daniel Bork, Yaron Orenstein, Ron Shamir, and Carl Kingsford. Improving the performance of minimizers and winnowing schemes. *Bioinformatics*, 33(14):i110–i117, 2017. doi:10.1093/BIOINFORMATICS/BTX235.

16 Malick Ndiaye, Silvia Prieto-Baños, Lucy M Fitzgerald, Ali Yazdizadeh Kharrazi, Sergey Oreshkov, Christophe Dessimoz, Fritz J Sedlazeck, Natasha Glover, and Sina Majidian. When less is more: sketching with minimizers in genomics. *Genome Biology*, 25(1):270, 2024.

17 Melissa E. O'Neill. Pcg: A family of simple fast space-efficient statistically good algorithms for random number generation. Technical Report HMC-CS-2014-0905, Harvey Mudd College, Claremont, CA, September 2014.

18 Yaron Orenstein, David Pellow, Guillaume Marçais, Ron Shamir, and Car Kingsford. Designing small universal k-mer hitting sets for improved analysis of high-throughput sequencing. *PLoS Computational Biology*, 13(10):e1005777, 2017. doi:10.1371/JOURNAL.PCBI.1005777.

19 David Pellow, Lianrong Pu, Bariş Ekim, Lior Kotlar, Bonnie Berger, Ron Shamir, and Yaron Orenstein. Efficient minimizer orders for large values of k using minimum decycling sets. *Genome Research*, 33(7):1154–1161, 2023.

20 Michael Roberts, Wayne Hayes, Brian R. Hunt, Stephen M. Mount, and James A. Yorke. Reducing storage requirements for biological sequence comparison. *Bioinformatics*, 20(18):3363–3369, July 2004. doi:10.1093/bioinformatics/bth408.

21 Kristoffer Sahlin. Effective sequence similarity detection with strobemers. *Genome Research*, 31(11):2080–2094, 2021. doi:10.1101/gr.275648.121.

22 Saul Schleimer, Daniel S. Wilkerson, and Alex Aiken. Winnowing: local algorithms for document fingerprinting. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, SIGMOD '03, pages 76–85, New York, NY, USA, 2003. Association for Computing Machinery. doi:10.1145/872757.872770.

23 Hongyu Zheng, Carl Kingsford, and Guillaume Marçais. Improved design and analysis of practical minimizers. *Bioinformatics*, 36(Supplement_1):i119–i127, 2020. doi:10.1093/BIOINFORMATICS/BTAA472.

24 Hongyu Zheng, Guillaume Marçais, and Carl Kingsford. Creating and using minimizer sketches in computational genomics. *Journal of Computational Biology*, 30(12):1251–1276, 2023. doi:10.1089/CMB.2023.0094.


# A Appendix

## A.1 Computing the exact density of random binary 10-minimizers

To compute the density of a random binary 10-minimizer with parameters $k, w$, it suffices to find, for each $(w+k)$-window containing at least one 10-$k$-mer, the probability of being charged (the case of windows with no 10-$k$-mers is covered by Lemma 1). For a $(w+k)$-window $v$ with $m$ distinct $k$-mers, this probability can be expressed as

$$\frac{[k\text{-prefix of } v \text{ is a 10-}k\text{-mer}] + [k\text{-suffix of } v \text{ is unique and is a 10-}k\text{-mer}]}{m}$$

The efficient enumeration of all binary $(w+k)$-windows is organized as follows.

**(1)** The set of all $(w+k)$-windows is viewed as a binary *prefix tree*; its nodes are all binary strings of length $\leq w+k$, and its edges are all pairs $(u, ua)$, where $a \in \{0,1\}$. In particular, the leaves of this tree are exactly all the $(w+k)$-windows.

**(2)** The prefix tree is processed by a recursive DFS; each node $za$ gets from its parent $z$ the set of 10-$k$-mers occurring in $z$ and the flag indicating whether the $k$-prefix of $z$ is a 10-$k$-mer. Hence, if $za$ is a leaf, the probability of $za$ being charged is computed in $O(1)$ operations.

**(3)** To speed up the DFS traversal, the recursive routine is called for a node $z$ only if the $k$-suffix of $z$ is a 10-$k$-mer (and thus may change the set of occurred 10-$k$-mers inherited from the calling ancestor). If no descendants of a node $z$ have 10-$k$-mers as suffixes, then all leaves in the subtree of $z$ have the same probability of being charged; this probability is computed at $z$ and multiplied by the number of those leaves.

We computed the densities of random binary 10-minimizers with $k = 7$ and $k = 12$ over ranges of $w$ values and compared these results to the estimate $\mathcal{R}^*(2, k, w)$ by Theorem 5. They are presented in Table A1, normalized to density factors. In all cases, the error of the estimate $\mathcal{R}^*(2, k, w)$ is less than 0.11% for $k = 7$ and less than 0.0022% for $k = 12$. For the same ranges, the maximum error of the density estimate for *random minimizers* is higher: greater than 0.25% for $k = 7$ and greater than 0.0036% for $k = 12$. This confirms that our density estimates for random 10-minimizers are good for the practical ranges of parameters.

**Table A1** The exact vs. estimated (by Theorem 5) values of the expected density factor for random 10-minimizers. For comparison, the exact and estimated values of the expected density factor of random minimizers are provided for the same ranges.

**(a)** $\sigma = 2$, $k = 7$.

| | Random 10-minim. | | Random minim. | |
|---|---|---|---|---|
| $w$ | exact | estim. | exact | estim. |
| 5 | 1.711426 | 1.710938 | 2.005127 | 2.000000 |
| 6 | 1.749573 | 1.750000 | 2.005025 | 2.000000 |
| 7 | 1.780111 | 1.779134 | 2.003988 | 2.000000 |
| 8 | 1.801437 | 1.801428 | 2.003471 | 2.000000 |
| 9 | 1.820292 | 1.819305 | 2.002756 | 2.000000 |
| 10 | 1.834556 | 1.834117 | 2.002331 | 2.000000 |
| 11 | 1.848158 | 1.846664 | 2.001902 | 2.000000 |
| 12 | 1.858888 | 1.857462 | 2.001677 | 2.000000 |
| 13 | 1.868592 | 1.866860 | 2.001520 | 2.000000 |
| 14 | 1.877106 | 1.875114 | 2.001489 | 2.000000 |

**(b)** $\sigma = 2$, $k = 12$.

| | Random 10-minim. | | Random minim. | |
|---|---|---|---|---|
| $w$ | exact | estim. | exact | estim. |
| 10 | 1.832525 | 1.832504 | 2.000072 | 2.000000 |
| 11 | 1.845776 | 1.845739 | 2.000054 | 2.000000 |
| 12 | 1.856962 | 1.856936 | 2.000044 | 2.000000 |
| 13 | 1.866605 | 1.866564 | 2.000031 | 2.000000 |
| 14 | 1.874966 | 1.874949 | 2.000022 | 2.000000 |
| 15 | 1.882349 | 1.882328 | 2.000013 | 2.000000 |
| 16 | 1.888892 | 1.888876 | 2.000006 | 2.000000 |
| 17 | 1.894742 | 1.894731 | 1.999999 | 2.000000 |
| 18 | 1.900005 | 1.899997 | 1.999994 | 2.000000 |
| 19 | 1.904763 | 1.904760 | 1.999989 | 2.000000 |
| 20 | 1.909094 | 1.909090 | 1.999985 | 2.000000 |
| 21 | 1.913039 | 1.913043 | 1.999981 | 2.000000 |
| 22 | 1.916660 | 1.916666 | 1.999978 | 2.000000 |
| 23 | 1.919991 | 1.920000 | 1.999975 | 2.000000 |
| 24 | 1.923064 | 1.923077 | 1.999972 | 2.000000 |

## A.2 Density of binary spacers vs. binary ABB+ minimizers

We compared the densities of binary spacers and binary ABB+ orders for $w = 24$ and $k = 8, \ldots, 26$, estimating them on a random sample of $10^8$ $(w+k)$-windows for each pair $(k, w)$ (Supplementary Table A2). We note that the estimate of the expected density factor of a random 10-minimizer (Theorem 5), rounded to 6 decimal places, equals 1.923077 for all points in the analysed range.

**Table A2** Comparing the density factors of binary spacer and binary ABB+ minimizer for $w = 24$. The densities are estimated from a sample of $10^8$ random $(w + k)$-windows for each pair $(k, w)$.

| $k$ | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|
| $D_{\mathsf{spacer}}(2,k,24)$ | 1.744185 | 1.741076 | 1.741098 | 1.742237 | 1.742177 | 1.742321 | 1.742793 | 1.743476 | 1.742896 |
| $D_{\mathsf{ABB+}}(2,k,24)$ | 1.854068 | 1.857927 | 1.858477 | 1.860609 | 1.858185 | 1.860894 | 1.860402 | 1.861672 | 1.861706 |

| $\cdots$ | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 |
|---|---|---|---|---|---|---|---|---|---|---|
| $\cdots$ | 1.743550 | 1.743088 | 1.742920 | 1.742048 | 1.744163 | 1.741836 | 1.742611 | 1.742745 | 1.742599 | 1.743465 |
| $\cdots$ | 1.862441 | 1.861741 | 1.861548 | 1.860859 | 1.860694 | 1.859866 | 1.861647 | 1.860313 | 1.858494 | 1.859848 |

## A.3 A comment on mod-minimizers

[10, Def. 7] defines *mod-sampling scheme* with the parameters $(\sigma, k, w)$ as follows: an internal parameter $t \leq k$ and a (pseudo)random linear order $\rho$ on $\Sigma^t$ are chosen; in each window $v$, the leftmost position $x$ of the $\rho$-minimal $t$-mer is computed, and the $k$-mer at position $x$ mod $w$ is sampled. (The positions in any string are numerated starting with 0.) We note that the case $t = k$ is trivial, as it implies $x$ mod $w = x$, and thus the scheme coincides with the "internal" minimizer $(\rho, w)$. So it may be safely assumed $t < k$.

[10, Lemma 13] claims that the mod-sampling scheme is a minimizer (called there *mod-minimizer*) if the parameter $t$ satisfies $t \equiv k \pmod{w}$. Let us give a counterexample.

Suppose there exist two $k$-mers $u_1$ and $u_2$, sharing a common $t$-prefix $z$, and two windows: $v_1$ begins with $u_1$ and contains $u_2$, while $v_2$ begins with $u_2$ and contains $u_1$. As the order $\rho$ of $t$-mers is random, we can assume that $z$ is the $\rho$-minimal $t$-mer in both $v_1$ and $v_2$. Then the mod-sampling scheme samples the $k$-prefix in both $v_1$ and $v_2$. However, no minimizer can sample this way: if $u_1$ has smaller (respectively, larger) rank than $u_2$, then the sampling for $v_2$ (respectively, for $v_1$) would be invalid. A particular example follows.

Let $\sigma = 2, w = 8, t = 7, k = 15$. The positions of $u_2$ in $v_1$ and of $u_1$ in $v_2$ are indicated in boldface. The example can be easily generalized to $k = 2i + 7$ for any $i \geq 1$.

$$
\begin{aligned}
u_1 &= 010101010\ 010101\\
u_2 &= 0101010\ 01010101\\
v_1 &= u_1 0101010 = 01\mathbf{0101010\ 0101010}1010\\
v_2 &= u_2 0010101 = 0101010\mathbf{10101010\ 010101}
\end{aligned}
$$

The conclusion is that (i) the condition $t \equiv k \pmod{w}$ does not guarantee that the mod-sampling scheme is a minimizer, and (ii) currently there is no class of mod-sampling schemes with $t < k$ that is **proved** to consist of minimizers.

## A.4 Calculating density of GreedyMini orders

To measure the density of GreedyMini orders, we conducted measurements using a random DNA sequence of $10^7$ nucleotides, or incorporated densities reported in the GreedyMini study [6]. The density values were measured as follows:

**(1)** **For $k = 12$ and $w \in [3, 100]$:** We utilized the density values reported in the GreedyMini study. Additionally, we evaluated published GreedyMini orders for $(k, w) \in \{(12, 20), (12, 30)\}$ across smaller $w$ values. For each parameter set, we reported the minimum density observed between the original published results and our supplementary measurements.

**(2)** **For $k = 24$ and $w \in [3, 100]$:** GreedyMini densities were omitted for this range, as this $k$ value exceeds the recommended $k$ range for GreedyMini.

(3) **For $w = 12$ and $k \in [3, 32]$:** We utilized the densities reported in the GreedyMini study. For $k$ values not explicitly covered in the original publication, we estimated the density by applying the $k$-extension with lexicographic tie-breaking to the published GreedyMini order constructed for the largest available $k$ and $w = 12$.

(4) **For $w = 24$ and $k \in [3, 32]$:** In the absence of published GreedyMini orders for these specific parameters, we calculated the density by applying the following orders (using $k$-extensions as needed) to measure the density for the setting of $(k, w) = (k', 24)$:
- All $(k', 15)$ orders for $3 \leq k' \leq 14$.
- The $(15, 17)$ orders for $15 \leq k' \leq 32$.
- The $(15, 30)$ orders for $15 \leq k' \leq 32$.
- The $(8, 19)$ orders for $8 \leq k' \leq 32$.

For every $(k, w)$ configuration, we report the lowest density observed across all tested orders.

## A.5 Key-retrieval times for the open-closed syncmer vs. miniception

The pseudocode of open-closed syncmer suggests it is theoretically slower than miniception due to an additional `if`. However, our benchmarking demonstrated fluctuating performance. Specifically, for $k = 16$, miniception was faster in half of the experimental runs for `std::hash`, while open-closed syncmer outperformed it in the other half. These differences, though alternating in direction, remained statistically significant per each run ($p < 10^{-9}$, Wilcoxon rank-sum test). After aligning the $s$-mer rank structures to memory addresses divisible by 64, both schemes were slower, yet miniception consistently outperformed open-closed syncmer, hinting at variability due to memory layout.

# Construction of Distinct $k$-mer Color Sets via Set Fingerprinting

**Jarno N. Alanko**
Department of Computer Science, University of Helsinki, Finland

**Simon J. Puglisi**
Department of Computer Science, University of Helsinki, Finland

## Abstract

The colored de Bruijn graph model is the currently dominant paradigm for indexing large microbial reference genome datasets. In this model, each reference genome is assigned a unique *color*, typically an integer id, and each $k$-mer is associated with a *color set*, which is the set of colors of the reference genomes that contain that $k$-mer. This data structure supports a variety of *pseudoalignment* algorithms, which aim to determine the set of genomes most compatible with a query sequence.

In most applications, many distinct $k$-mers are associated with the same color set. In current indexing algorithms, color sets are typically deduplicated and compressed only at the end of index construction. As a result, the peak memory usage can greatly exceed the size of the final data structure, thereby requiring the use of slower external-memory construction algorithms that require vast amounts of disk space.

In this work, we present a Monte Carlo algorithm that constructs the set of distinct color sets for the $k$-mers directly in any individually compressed form. The method performs on-the-fly deduplication via incremental fingerprinting. We provide a strong bound on the error probability of the algorithm, even if the input is chosen adversarially, assuming that a source of random bits is available at run time. We show that given an SBWT index of 65,536 *S. enterica* genomes, we can enumerate and compress the distinct color sets of the genomes to 40 GiB on disk in 7 hours and 17 minutes using 32 threads, requiring only 14 GiB of RAM and no temporary disk space, with an error probability of at most $2^{-82}$.

**2012 ACM Subject Classification** Applied computing → Bioinformatics

**Keywords and phrases** k-mer, SBWT, BWT, inverted index, genomics, fingerprinting, deduplication

**Digital Object Identifier** 10.4230/LIPIcs.WABI.2026.6

**Supplementary Material** *Software (Source code)*: `https://github.com/jnalanko/themisto2`
archived at `swh:1:dir:31347e19750df797ba710299cc7d6f701ed0e144`
*Software (Source code)*: `https://github.com/jnalanko/color_set_fingerprinting_experiments`
archived at `swh:1:dir:ecc5d1b6dc2468cc1128686da7bd978757a5f7b6`

**Funding** This work was supported in part by the Research Council of Finland through grants 339070 and 370538.

## 1 Introduction

Indexing large collections of microbial reference genomes to support similarity search is now a widespread task in modern genomics, with the the colored de Bruijn graph model currently dominant paradigm [4, 12, 14]. In this model, each reference genome is assigned a unique color, and each $k$-mer is associated with a *color set*, that is, the set of colors of the references that contain that $k$-mer.

Early incarnations of this model used a binary matrix to represent color information [16, 20], where rows correspond to the k-mers of the dataset and the columns to colors (genomes). Cell $[i, j]$ of the matrix is 1 if and only if the $k$-mer (with rank) $i$ occurs in genome $j$. Rows of the matrix correspond to color sets. When stored naively, the matrix representation is


© Jarno N. Alanko and Simon J. Puglisi;
licensed under Creative Commons License CC-BY 4.0

large – a collection of 64 thousand Salmonella genomes with $k = 31$ results in a matrix of some 44 trillion cells. However, a particular property of the color matrix, especially manifest by pangenomes, is that many rows (i.e., color sets) are identical, a phenomenon borne out of the co-occurrence of $k$-mers in genomes. Identification and removal of duplicate rows is thus a key step in reducing space. Another important space-saving technique is to judiciously choose the representation used for individual color sets (those that remain after duplicate detection) according to the density of the color set, an approach borrowed from the search engine literature [23]. Another important reason to compress the matrix row-wise (as opposed to column-wise) is that important applications, such as pseudoalignment, which involve intersection or thresholded-union of colors sets, require fast access to *rows* of the matrix.

Ideally, we would like to arrive at a row-compressed representation of the matrix without having to realise the whole uncompressed matrix as an intermediate step. Various authors have made attempts at efficient (compressed) matrix construction. Metagraph [17] constructs the matrix column-wise, which is easy to implement, but loses the ability to deduplicate (or otherwise compress) rows until the whole matrix is in hand, thus requiring a huge amount of intermediate space. Another approach (adopted by, e.g., Bifrost [14], GGCAT [8]) is to use dynamic set data structures to construct the rows incrementally, but this makes parallelism complicated, and still makes the intermediate space significantly higher than the final space. Current methods deduplicate rows of $k$-mers that are in the same unitig, but deduplicating across unitigs during construction remains challenging.

In this paper, we present an algorithm that (1) parallelizes well, requiring only a certain well-supported set of atomic CPU instructions, and no higher-level synchronization primitives (2) deduplicates rows during construction, even across unitigs (3) does not require dynamic data structures: instead, a static data structure is constructed in multiple stages. We describe a Monte Carlo algorithm that constructs the set of distinct color sets for the $k$-mers directly in any individually compressed form, using the unitigs of each reference genome. The method performs on-the-fly deduplication via incremental fingerprinting. We provide a strong bound on the error probability of the algorithm, even if the input is chosen adversarially, assuming that a source of random bits is available at run time.

We show that given an SBWT index [3] of 65,536 *S. enterica* genomes, we can enumerate and compress the distinct color sets of the genomes to 40 GiB on disk in 7 hours and 17 minutes, using only 14 GiB of RAM and no temporary disk space. For context, the original genome files in FASTA format occupy 294 GiB on disk.

The remainder of this paper is structured as follows. In the next section we lay down notation and basic concepts used throughout. Then, in Section 3, we describe the main tenents of our approach, including details for its efficient implementation. In Section 4 we describe results of extensive experiments measuring performance. Conclusions and avenues for future work are then offered.

## 2 Notation and preliminaries

**Strings, $k$-mers and color sets.** A string of length $n$ is an array $T[0..n-1]$ with elements from an alphabet $\Sigma$. We use both closed and half-open interval notations for substrings, that is, the substring of $T$ starting from index $i$ and ending at index $j$, not including index $j$, is denoted with $T[i..j)$ or $T[i..j-1]$. In this paper, we assume the DNA alphabet $\Sigma = \{A, C, G, T\}$, though our algorithm generalizes to any alphabet. A *$k$-mer* is a string of length $k$. In the context of DNA, a $k$-mer is called *canonical* if it is lexicographically less or equal to its reverse complement on the opposite strand. The *$k$-spectrum*

of a string $T$, denoted with $S_k(T)$, is the set of distinct $k$-mers contained in $T$, that is, $S_k(T) = \{T[i..i+k) \mid 0 \leq i < |T| - k + 1\}$. The input to $S_k$ is also allowed to be a *list* of strings. For a list of strings $T_0, \ldots, T_{m-1}$, we define $S_k(T_0, \ldots, T_{m-1}) = \bigcup_{i=0}^{m-1} S_k(T_i)$. Given a list of strings $T_0, \ldots, T_{m-1}$, the *color set* of a $k$-mer $x \in S_k(T_0, \ldots, T_{m-1})$, denoted with $\mathsf{cs}(x, T_0, \ldots, T_{m-1})$, is the set of indices of input strings that contain $x$, that is, $\mathsf{cs}(x, T_0, \ldots, T_{m-1}) = \{i \in \{0, \ldots, m-1\} \mid x \text{ is a substring of } T_i\}$. When the input strings $T_0, \ldots, T_{m-1}$ are clear from the context, we drop them in the notation and write just $\mathsf{cs}(x)$.

**De Bruijn Graphs.** The node centric de Bruijn graph of a set of $k$-mers is a graph where each distinct $k$-mer in the set is a node, and there is an edge from $k$-mer $x$ to $k$-mer $y$ iff $x[1..k) = y[0..k-1)$. The de Bruijn graph of a list of strings $T_0, \ldots, T_{m-1}$ is the de Bruijn graph of their $k$-mers $S_k(T_0, \ldots, T_{m-1})$. A *unitig* is a maximal non-branching simple path $(v_0, \ldots, v_{\ell-1})$ in the de Bruijn graph in the following sense: for $i > 0$, the in-degree of $v_i$ is 1, and for $i < \ell - 1$, the out-degree of $v_i$ is 1, and the path cannot be extended in either direction without breaking these properties.

**Rank queries.** Given an array of bits $B[0..n)$, a rank-1 query at position $i$, denoted with $rank_1(B, i)$, counts the number of 1-bits in $B[0..i)$. Likewise, a rank-0 query, denoted with $rank_0(B, i)$ counts the number of 0-bits in $B[0..i)$. When we refer to just "rank queries", or omit the subscript 1 from the notation, we mean rank-1 queries. There exist practical data structures to support both types of rank queries on $B$ in $O(1)$ time using $o(n)$ bits of extra space [21].

## 2.1 State of the art in compressed color set construction

Most currently available tools build an uncompressed color set representation before compressing the data into the final form. For example, the recommended construction pipeline of Metagraph [17] is to first index the distinct $k$-mers without colors, then construct the color annotation of $k$-mers of each color independently, then combine the annotations and finally compress them using a variety of methods [18, 9, 5]. While the final space can be very small, the intermediate space during construction in memory or on disk can be orders of magnitude larger.

**Bifrost** [14] first constructs the unitigs of the input, indexes them with minimizer hashing, and then employs dynamic set data structures to construct the color sets incrementally. Space is saved by constructing the coloring at the level of *subunitigs*, that is, instead of associating $k$-mers to color sets, Bifrost breaks unitigs into runs of $k$-mers whose color set is the same, and colors the runs instead of individual $k$-mers. This provides color set deduplication within unitigs, but not *across* unitigs. The encoding of each color set is chosen to exploit properties like density and long runs of consecutive color identifiers. Parallel synchronization is managed with a system of mutex locks: instead of a single global lock, the data structure is striped with many locks so that each lock only locks one slice.

The **GGCAT** tool [8], which is at the heart of the color construction pipelines of Themisto [4] and Fulgor [12], constructs both the unitigs and the color sets of the $k$-mers at the same time by bucketing $k$-mers by minimizers and assembling to unitigs, and tracking color information at the same time. The color sets in unitig fragments are maintained as dynamic resizable vectors of integers encoded with a custom variable-length encoding inspired by Google Varint. This also provides color set deduplication within unitig fragments, but not across unitigs until the end. Finished color sets are inserted into a global hash table, and compressed using Lempel-Ziv encoding combined with Huffman coding. The intermediate working space remains significantly larger than the final compressed space.

**Cuttlefish 3** [19], described in a preprint and developed concurrently with the present work, uses an external memory construction algorithm with combineable hash fingerprints to track color-changing nodes. The algorithm is claimed to be 3–4 times faster than GGCAT, but generally has a higher RAM and disk usage. As with the other tools, the color sets are inserted into a concurrent dynamic data structure.

In general, using dynamic set data structures for this problem is not ideal since their space usage is usually significantly higher than their static counterparts, and they lead to memory fragmentation due to frequent reallocations. GGCAT 2.0 uses a custom dynamic memory allocator to mitigate this issue to some extent, but it can still use up to twice as much memory as needed. Another drawback of dynamic data structures is that they do not work well with parallelism. A synchronization mechanism is usually needed to prevent two threads from modifying the same structure at the same time, which results in a communication overhead between threads and wasted time due to threads having to wait for their turn to access the data.

## 2.2 Sparse-dense color set representation

The goal of our work is to build a compact representation of the distinct color sets of the $k$-mers of the input genomes. There is a growing literature on ways to compress the color sets (see [6] and references therein), much of it stemming from literature on inverted index compression [23]. In this work, our target is the representation from Themisto [4] (Fulgor [12] and Bifrost [14] employ similar schemes). Since our algorithm depends on details of the representation, we now recall the details of the representation here.

Let $d$ be the number of distinct color sets. Each color set is encoded either in sparse or dense form, depending on which representation requires less space (this is defined precisely in a moment). Let $B$ be a bit vector of length $d$ such that $B[i] = 1$ if color set $i$ is stored in sparse form, and $B[i] = 0$ if it is stored in dense form. The bit vector $B$ is indexed for rank queries. A sparse set is stored as a sorted list of its elements, encoded using $\lceil \log m \rceil$ bits per element. The representations of all sparse sets are concatenated into a single array $A$ of $\lceil \log m \rceil$-bit integers. The starting points of each set are stored in an offset array $O$ of 64-bit integers, where $O[j]$ gives the starting position of the $j$-th sparse set. A sparse set of $s$ elements therefore occupies $\lceil \log m \rceil s + 64$ bits in total. Dense color sets are stored as bitmaps of length $m$ bits. All the bitmaps are concatenated into a single bitmap $D$. Since all dense sets have the same size, no offset array is required, and each dense set occupies exactly $m$ bits. A color set of size $s$ is stored in sparse form if $\lceil \log m \rceil s + 64 < m$, and in dense form otherwise. To access color set $i$, we first inspect $B[i]$. If $B[i] = 1$, then its rank among sparse sets is $j = rank_1(B, i)$, and its elements are stored starting at offset $O[j]$ in $A$. If $B[i] = 0$, then its rank among dense sets is $r = rank_0(B, i)$, and its bitmap starts at bit index $rm$ in $D$.

# 3 Algorithm

## 3.1 Problem statement and setup

To simplify our exposition, we model a genome as a *single* string from the DNA alphabet. The input to our algorithm is a set of $m$ genomes $\mathcal{G} = \{G_0, \ldots, G_{m-1}\}$. Multi-string genomes can be easily supported in implementation. The goal is to construct the set of distinct color sets of $\mathcal{G}$, and an index structure that can quickly return the color set of any query $k$-mer.

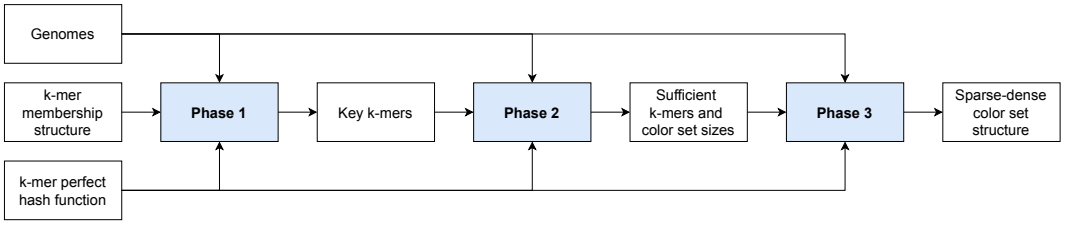



**Figure 1** Algorithm overview. The $k$-mer membership structure can be discarded after Phase 1.

We assume we have data structure support for de Bruijn graph operations to (1) compute the outdegree of a node (2) compute the indegree of a node (3) move from a node to any of its in-neighbors. These can be provided with any $k$-mer lookup structure, e.g. with the SBWT or the SSHash index, or any hashing-based compact de Bruijn graph data structure.

Further, we assume that we have an injective (i.e. perfect) hash function $h : S_k(\mathcal{G}) \to \{0, \ldots, n-1\}$ mapping the distinct $k$-mers to distinct integers. The largest hash value $n-1$ is allowed to be greater than $|S_k(\mathcal{G})|$, but should be reasonably small to obtain good space efficiency. Examples of such functions include SSHash [22], which can be used to map a $k$-mer to its starting position in the unitigs of the input genomes, and SBWT [3], which can be used to map each $k$-mer to its *colexicographic rank* among the input $k$-mers (roughly speaking). Any generic perfect hash function, e.g. PtrHash [13] or Recsplit [11], will also do.

## 3.2 Algorithm

We introduce the notion of a *color-set covering $k$-mer set*. Given a list of genomes $\mathcal{G} = G_0, \ldots, G_{m-1}$ (strings), a color set covering $k$-mer set is a set of $k$-mers $A \subseteq S_k(\mathcal{G})$ such that every color set is found at least once. More formally:

▶ **Definition 1** (Color-set covering $k$-mer set). *Given a set of strings $\mathcal{G} = G_0, \ldots, G_{m-1}$, a set of $k$-mers $A \subseteq S_k(\mathcal{G})$ is a color-set covering $k$-mer set if for every $x \in S_k(\mathcal{G})$ there exists a $k$-mer $y \in A$ with $\mathsf{cs}(x, \mathcal{G}) = \mathsf{cs}(y, \mathcal{G})$.*

Finding a small color-set covering $k$-mer set quickly is a major piece of our approach. We proceed in three phases. **In the first phase**, we find a color-set covering $k$-mer set $C_1 \subseteq S_k(\mathcal{G})$, which may not be minimal, but tends to be much smaller than $S_k(\mathcal{G})$ because it exploits the property that in genomic data, the color sets of $k$-mers in the same unitig tend to be the same. We call the $k$-mers in $C_1$ *key $k$-mers*. **In the second phase**, we use a fingerprinting scheme to shrink this set to obtain a *minimal* color-set covering $k$-mer set $C_2 \subseteq C_1$, containing exactly one $k$-mer per distinct color set in the data, called the *sufficient $k$-mers*. **In the third phase**, we construct the color sets of $k$-mers in $C_2$ into a sparse-dense structure as defined in Section 2.2. See Figure 1 for a visual representation of the inputs and outputs to each phase. We now proceed to describe the phases in detail.

### Phase 1: Marking key $k$-mers

In this phase, we identify a set of *key $k$-mers*, such that every distinct color set is the color set of at least one key $k$-mer. A $k$-mer $x$ is key if any of the below holds:

1. $x$ is the last $k$-mer of a string in the input genomes.
2. $x$ has an out-neighbor that is the first $k$-mer of a string in the input genomes.
3. $x$ is the last $k$-mer of a unitig of the de Bruijn graph of all the $k$-mers, that is, $x$ has outdegree not equal to 1, or $x$ has an out-neighbor with in-degree greater than 1.

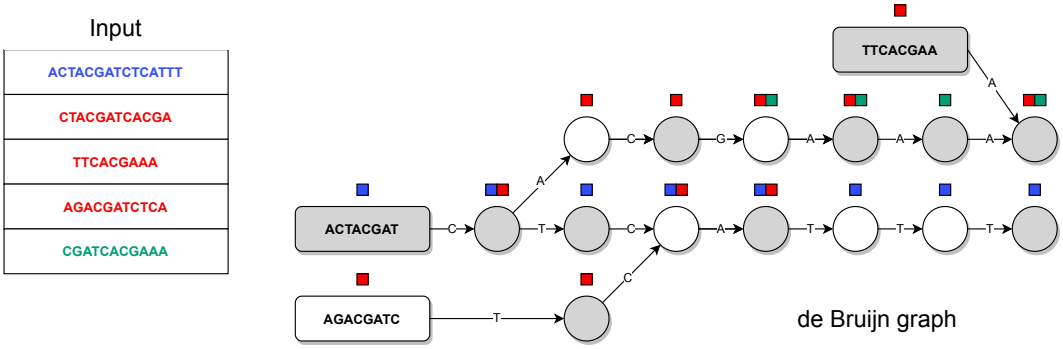



**Figure 2** Phase 1. Each node represents a $k$-mer, with $k = 8$. To keep the figure clean, the $k$-mer labels are written only for source nodes. The edge labels are the last characters of the destination $k$-mers. So-called *key* $k$-mers are the nodes with a gray background. The color sets of the nodes are shown above the nodes. The color set of each white node can be obtained by walking forward to the nearest gray node.

We initialize a bit vector of length $n$ to mark the perfect hash values of $k$-mers according to the hash function $h$ that match any of the three above conditions. To mark $k$-mers matching conditions 1 or 2, we read through the input genomes and apply $h$ on the last $k$-mer of each sequence, and the de Bruijn graph in-neighbors of the first k-mer of each sequence. Then, we use the de Bruijn graph structure to identify and mark all $k$-mers fulfilling condition 3.

If $x$ is not marked, it has exactly one out-neighbor $y$, and, moreover, $y$ has the same color set as $x$ (See Lemma 1 in [4]). We call $y$ the successor of $x$ and denote $y = \mathsf{succ}(x)$. Let $\mathsf{depth}(x)$ be the smallest $\ell$ such that $\mathsf{succ}^\ell(x)$ is marked, where $\mathsf{succ}^\ell(x)$ is the successor function iterated $\ell$ times. Our construction guarantees that such $\ell$ always exists, as shown by the following lemma.

▶ **Lemma 2.** *Every $k$-mer $x$ has a finite* $\mathsf{depth}(x)$.

**Proof.** For any marked $k$-mer $x$, we have $\mathsf{depth}(x) = 0$. If a node is not marked, it has a unique successor. We need to show that iteratively applying $\mathsf{succ}(x)$ eventually leads to a marked node. For a contradiction, let us assume this does not happen. Then, the sequence of successors eventually gets stuck in a cycle of non-marked nodes. By condition 3, all nodes on the cycle have both in-degree and out-degree 1. This means that there must exist a marked node on the cycle because the $k$-mers on the cycle come from some original sequence, whose last $k$-mer is marked, and that $k$-mer is somewhere on the cycle. ◀

▶ **Lemma 3.** *For every distinct color set in the input, there is at least one marked $k$-mer that has that color set.*

**Proof.** We prove the claim by using induction on $\mathsf{depth}(x)$. Suppose the color sets of all $k$-mers with $\mathsf{depth}(x) \leq \ell$ are covered, and let $f(x)$ be any key $k$-mer whose color set is the same as that of $x$. For the base case, at $\ell = 0$ the claim holds because nodes with depth 0 are marked, and we can set $f(x) = x$. Consider $k$-mers with $\mathsf{depth}(x) = \ell + 1$. Since $\mathsf{depth}(x) > 0$, they each have a unique successor with the same color set, whose color set is $\mathsf{cs}(f(\mathsf{succ}(x)))$, which is correct by induction since $\mathsf{depth}(\mathsf{succ}(x)) = \ell$. ◀

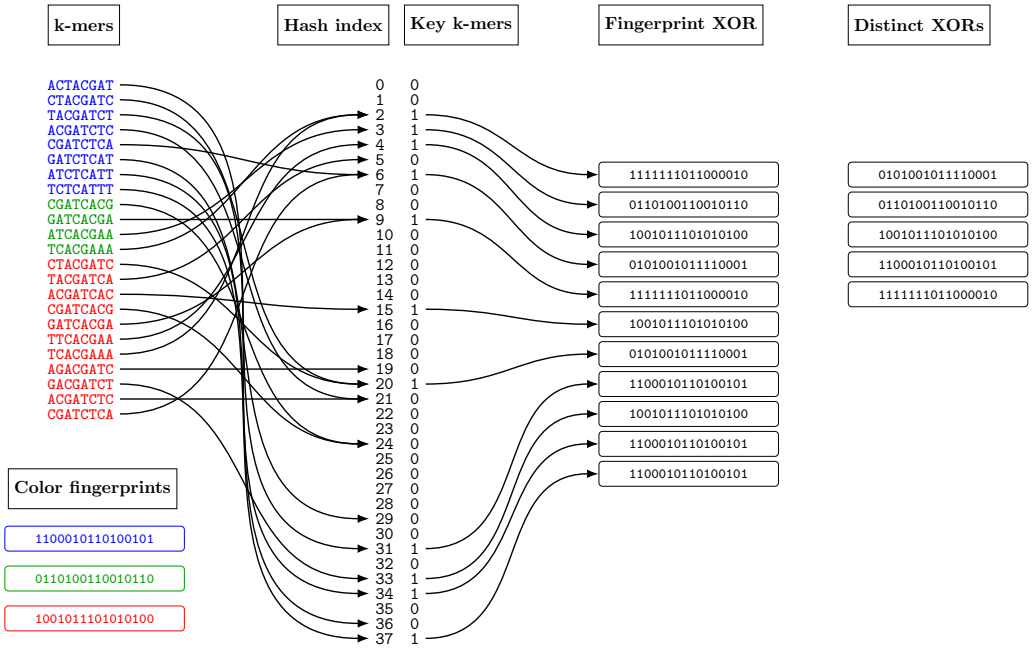



**Figure 3** Fingerprinting in Phase 2. At the start of the phase, each genome (color) is associated with a random $\ell$-bit fingerprint. In this picture, $\ell = 16$. Each $k$-mer is hashed with the perfect hash function, and in the event it turns out to be a key $k$-mer, the fingerprint of the color is xorred into the aggregate fingerprint of the set. In the end, the aggregate fingerprints of the sets are sorted and deduplicated.

## Phase 2: Computing fingerprints and marking sufficient $k$-mers

In this phase, we compute a fingerprint of the color set of every key $k$-mer marked in the first phase, and in the end, assign ids $0, 1 \ldots$ to distinct fingerprints. The input is the key $k$-mer mark bit vector from Phase 1, the input genomes $\mathcal{G}$, and a perfect hash function on the $k$-mers of $\mathcal{G}$. The output is a bitvector marking sufficient $k$-mers, and an array containing the sizes of the color sets of the sufficient $k$-mers in the order they appear in the bitvectors.

The fingerprinting scheme is an instance of *tabulation hashing*, a generic technique dating back to 1970, originally used for fingerprinting distinct chess positions [24]. First, we assign a random $\ell$-bit fingerprint for each distinct individual color, and store these in a table. The higher the value of $\ell$, the smaller the probability of error in the algorithm will be. We denote the chosen fingerprint of color $c$ with $f(c)$. The fingerprint of a *set* of colors will be the xor of the fingerprints of the individual colors in the set. We denote the fingerprint of set $A$ with $F(A)$, that is, $F(A) = \bigoplus_{c \in A} f(c)$. This manner of fingerprinting lends itself to very clean collision analysis. The next lemma shows that $F$ is a universal hash function family over the random choice of $f$.

▶ **Lemma 4.** *Let $A$ and $B$ be two distinct sets of colors and select the function $f$ uniformly at random from all functions $f : \mathbb{Z}_m \to \mathbb{Z}_{2^\ell}$, where $m$ is the total number of possible colors. We have* $\Pr(F(A) = F(B)) = 2^{-\ell}$.

**Proof.** Since $A \neq B$, there exists a color $c$ that occurs in exactly one of $A$ and $B$. Without loss of generality, suppose that $c \in A$ and $c \notin B$. We have that $F(A) = F(B)$ if and only if $F(A \setminus \{c\}) \oplus f(c) = F(B)$, where $\oplus$ is the xor operation. Rearranging, this means

$f(c) = F(A \setminus \{c\}) \oplus F(B)$. There are $2^{\ell m}$ possible functions $f$. To count the functions for which the equation holds, first choose a value for $f(x)$ for each $x \neq c$. There are $2^{\ell(m-1)}$ possibilities. This determines the value of $F(A \setminus \{c\}) \oplus F(B)$, so there is exactly one way to choose $f(c)$ to make the equation hold. Hence, the fraction of functions for which the equation holds is $2^{\ell(m-1)}/2^{\ell m} = 2^{-\ell}$. ◀

This gives us the following bound on the probability of a hash collision across all given color sets.

▶ **Lemma 5.** *Given a set of distinct sets $A_0, \ldots, A_{N-1}$, the probability that there exists two sets $A_i \neq A_j$ such that $F(A_i) = F(A_j)$ is at most $N^2/2^{\ell+1}$.*

**Proof.** By the union bound, and Lemma 4, the probability is at most $\binom{N}{2}2^{-\ell} = \frac{N(N-1)}{2^{\ell+1}}$. Rounding the numerator up to $N^2$ gives the claimed bound. ◀

For example, for $\ell = 128$ and $N = 10^9$, we have a collision probability of at most $10^{18}/2^{129} \approx 1.47 \cdot 10^{-21}$. This manner of fingerprinting has the useful property that the fingerprint of a set can be built incrementally by initializing the fingerprint to a $\ell$-bit string of zeroes, and individually xorring the fingerprints of the set's colors in any order.

To build the fingerprints, we initialize an array $A[0, s-1]$ of $\ell$-bit integers, where $s$ is the number of key $k$-mers identified in the first phase. We preprocess the key $k$-mer marks bit vector $B$ from Phase 1 for rank queries, so that the fingerprint of $k$-mer $x$ is at $A[\mathsf{rank}(B, h(x))]$. Then, we iterate over all the $k$-mers in each input genome, and for each key $k$-mer, xor the fingerprints of the colors of the genome to the corresponding indexes in $A$. See Figure 3.

This can be done in parallel using atomic instructions to apply the xor-operations to elements of $A$. Since xor is commutative, no further synchronization between threads is required because any order of updates results in the same final result.

At the same time as building $A$, we also build another array $E$ of the same length, where $E[i]$ stores the number of distinct elements xorred into $A$. The arrays $E$ can also be built in parallel with widely supported atomic increment operations.

We obtain the set of distinct fingerprints by sorting and deduplicating in-place. Let $d$ be the number of distinct fingerprints found. The set of distinct fingerprints are assigned ids from $0, \ldots, d-1$. We pick for each distinct color set a single representative $k$-mer that has that color set. The representative among $k$-mers that have a given color set is defined as the one which has the smallest hash value according to $h$. The representatives form the set of sufficient $k$-mers for this dataset. We mark these $k$-mers in a bit vector $B_2$ of length $n$, and index the bit vector for rank queries.

In the final phase, we build the color sets of the representative $k$-mers.

## Phase 3: Building the sparse-dense structure

In this phase, we build the sparse-dense structure described in Section 2.2 for the sufficient $k$-mers marked in the previous phase. Let $B_2$ be the bit vector marking the sufficient $k$-mers from the previous phase. We start by allocating a "skeleton" of the final sparse-dense data structure, containing the bit vector that marks sparse sets, the space for the concatenation of dense bitmaps and sparse integer lists, and the offset array, all described in Section 2.2. The marking bit vector is allocated and populated first based on the sizes of the sets, which informs us how many bits are required for the sparse and dense components. To fill in the elements of the sets, we run one final pass over the input genomes, applying hash function $h$

on each $k$-mer $x$, and whenever $B_2[h(x)] = 1$, we query the skeleton to find where the set is stored in the data structure, and add the element to the set. Some care is required to append to the same sets from multiple threads. These details are explained toward the end of the next subsection.

## 3.3 Implementation details and space analysis

**Phase 1.** Let $n$ be the number of distinct $k$-mers in the input. Phase 1 requires a set membership structure on the $k$-mers in order to measure indegrees and outdegrees of $k$-mers in the de Bruijn graph, and to traverse to neighboring $k$-mers. We also require a perfect hash function on the $k$-mers. The SBWT [3] and the SSHash [22] data structures both double as a $k$-mer membership structure and a perfect hash function: for SBWT, the hash value is the position of the $k$-mer on the SBWT, and in SSHash it is the starting point of the $k$-mer in the concatenation of the unitigs of the data. These data structures typically take $2.5 - 15$ bits per $k$-mer on real-world sequencing data [3, 22]. In the worst case, up to $k \log \sigma$ bits are required, where $\sigma$ is the size of the alphabet – this represents the case where the $k$-mers do not overlap at all, so we cannot do better than encoding them individually using $\log \sigma$ bits per character. The key $k$-mer marks take another $n$ bits, where $n$ is the size of the range of the perfect hash function. In summary, the space used in phase 1 is $\mathsf{kmers} + n$, where $\mathsf{kmers}$ is the combined space of the $k$-mer membership structure and the perfect hash function into $\{0, \ldots, n-1\}$, with practical values ranging from $3.5 - 16$ bits per distinct $k$-mer. In this work, we use the SBWT as a perfect hash function, but we emphasize this is not a fixed choice.

**Phase 2.** In Phase 2, we retain the key $k$-mer bit vector $B$, but we no longer need the set membership structure of $k$-mers, only the perfect hash function. Let $n_{key}$ be the number of key $k$-mers and $\ell$ be the fingerprint length in bits. Since each unitig introduces only at most one key $k$-mer, and each input sequence introduces at most two, we can bound $n_{key} \leq u + 2r$, where $u$ is the number of unitigs and $r$ is the number of input sequences. The fingerprint array takes $n_{key}\ell$ bits. To deduplicate the fingerprints in the end and assign ids and sizes to unique color set ids, we build and sort an array of triples (set fingerprint, key $k$-mer hash value, set size), using the set fingerprint as the primary sort key, breaking ties using the key $k$-mer hash value. The fingerprints take $\ell$ bits each, the hash value takes $\lceil \log_2 n \rceil$ bits and the set size $\lceil \log_2 m \rceil$ bits, so in total, the triples take $n_{key}(\ell + \lceil \log_2 n \rceil + \lceil \log_2 m \rceil)$ bits, (though in practice to simplify code, we use 64-bit integers for the second and third elements of triples). We assign integer identifiers $0, 1, \ldots$ for the distinct set fingerprints in the sorted order. The smallest key $k$-mer that has each distinct fingerprint is chosen as the representative $k$-mer for that color set. These are marked in the bit vector $B_2$ of length $n$. In summary, all these structures together take $\mathsf{PHF} + n_{key}(\ell + \lceil \log_2 n \rceil + \lceil \log_2 m \rceil) + 2n + o(n)$, with $n_{key} \leq u + 2r$, where $\mathsf{PHF}$ is the size of the perfect $k$-mer hash function, the $o(n)$ term is for the rank structures on the key $k$-mer and sufficient $k$-mer marks.

**Phase 3.** We retain from the previous phase the perfect hash function, the bit vector $B_2$ marking sufficient $k$-mers, and the array of color set sizes. We build the sparse-dense color set representation described in Section 2.2 directly. Since we know the size of each distinct set, we are able to pre-allocate all the required memory: the bit vector that marks sparse sets, the concatenation of the sparse sets, the offset array to the sparse sets, and the concatenation of bitmaps of dense sets. All this takes $\lceil \log m \rceil s + 64 d_{sparse} + d_{dense} m + d + o(d)$ bits, where $d$ is the number of distinct color sets, $d_{sparse}$ is the number of sparse sets and $d_{dense}$ the number

of dense sets ($d_{sparse} + d_{dense} = d$), and $s$ is the total number of elements in all sparse sets. The overhead on top of this is the size of the perfect hash function, the 64-bit color set sizes array and the sufficient $k$-mers bit vector, for a total overhead of $\mathsf{PHF} + 64d + n + o(n)$.

**Parallelism in phase 3.** In this phase, we build the sets by adding elements one by one, but this makes parallelism nontrivial since two threads might try to add elements to the same set at once. Incidentally, the construction algorithm in Bifrost [14] runs into the same issue, which is solved there by using mutexes to ensure that no two threads operate on the same set at once. Our construction method and color set representation allows for a completely lock-free solution. Updating the dense bitmaps in parallel is simple since modern CPUs support atomically setting a single bit in a word. Updating sparse sets is more complicated because each thread needs to *append* to the same list. However this can be handled by storing the offset where the next item should be stored, using an atomic fetch-increment operation to read the offset, and then write to the fetched offset. The atomic fetch-increment ensures that no two threads append to the same position. The write itself is also tricky since we are not operating on full memory words but rather $\lceil \log m \rceil$ bit integers. However the write can be achieved atomically by zeroing the target bit region with atomic and-operations, and writing the bits with atomic or-operations – no two different threads ever modify the same bits at once, so all interleavings of the parallel operations will result in the same final result.

**Running from input genomes with repeated $k$-mers.** The fingerprinting assumes that every color fingerprint is xorred into the aggregate color set fingerprint only once. Otherwise, if an element is xorred an even number of times, it is the same as not xorring at all, and the collision analysis in Lemma 4 breaks. There is no easy fix for this, since if we could detect whether a given color has been added to a given set, it would amount to having a dynamic set membership data structure for each of the color sets being constructed, which is difficult to implement in small space – in fact, this type of a scheme is precisely what we aim to avoid with the fingerprinting. Instead, when running from genomes that might have duplicate $k$-mers, we add a deduplicating layer before xorring to avoid xorring the same color twice. We implement this by maintaining the set of perfect hash values of $k$-mers that have been processed so far for the current genome. This set is stored as a hash table, until the space of the hash table becomes bigger than $n$, at which point we switch to an indicator bit vector of length $n$ marking the perfect hash values of those $k$-mers that have been processed. Each deduplicating structure is maintained in memory only for the duration of the processing of its corresponding genome. The extra space for this is manageable if the input sequences are short, like individual microbial genomes. Otherwise, the input sequences can be preprocessed into unitigs to remove duplicates, so that no deduplication is required during construction.

**Constructing directly to disk.** To decrease the peak working space, we can construct the final sparse-dense data structure directly on disk. This is possible because after phase 2, we know in advance the size and the preferred representation of each set. We can thus allocate all the space for the final data structure on disk up front. Then, we split the input genomes into $p \leq m$ chunks (e.g. $p = 8$), and process each chunk of genomes in turn. Chunk $i$ will contain the colors in the range $[i\lceil m/p \rceil .. \min((i+1)\lceil m/p \rceil, m))$. For each chunk, we run phase 3 to build a sparse-dense structure for the colors of the chunk, and then stream over the file on disk, writing the constructed colors to their memory locations in the final data structure. This way, only a fraction of approximately $1/p$ of the final sparse-dense data structure is held in memory at once. Overall, this means running $p$ passes over the output file, so $p$ should be kept reasonably small.

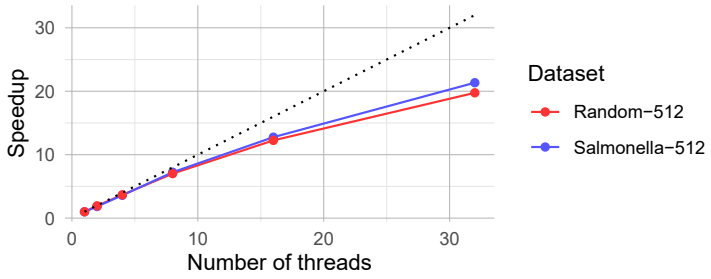



**Figure 4** The speedup achieved for index construction on 512 genomes of the random and Salmonella data sets as the number threads are increased ($k = 31$).

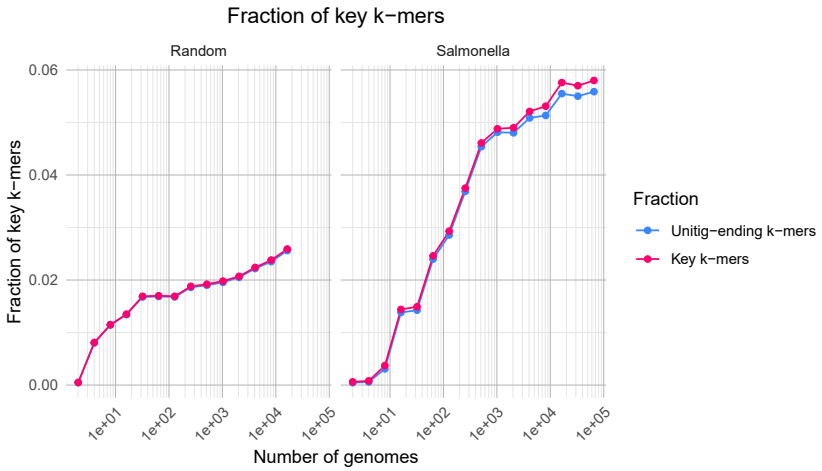



**Figure 5** Fraction of $k$-mers ($k = 31$) that are marked as key $k$-mers as a function of the number of genomes. The blue line shows $k$-mers that are at the ends of unitigs, and hence always key $k$-mers ($k = 31$).

## 4 Experiments

We implemented our method using an SBWT index to provide both $k$-mer lookup and the perfect hash function for the input $k$-mers: the hash value of a $k$-mer is its position in the SBWT. The queries are sped up by utilizing the longest common suffix array [1] to enable streaming $k$-mer lookup via the $k$-bounded streaming matching statistics algorithm [2]. Both DNA strands are indexed, and these data structures take roughly 10 bits per $k$-mer (20 bits per canonical $k$-mer). The implementation is a part of the pseudoalignment tool Themisto 2, available at `https://github.com/jnalanko/themisto2`. Scripts to reproduce the experiments are available at `https://github.com/jnalanko/color_set_fingerprinting_experiments`.

### 4.1 Hardware

All experiments were conducted on a single Linux server running Ubuntu 20.04 with kernel 5.15.0-153-generic. The system is equipped with 504 GiB of available RAM and an AMD Ryzen Threadripper PRO 3975WX processor (32 cores, up to 64 threads, 3.5 GHz clock, 32 × 32 KiB L1 cache, 32 × 512 KiB L2 cache, 128 MiB L3 cache). The storage consists of four 4 TB Seagate ST4000NM000A CMR HDDs, with observed[1] sustained write speeds of approximately 400 MiB/s.

[1] As reported by the command `dd if=/dev/zero of=out.bin bs=1MB count=100000 oflag=direct status=progress`.

**Table 1** Performance metrics for $n \geq 128$ genomes. The Salmonella datasets are labeled S-$n$ and the random datasets are labeled R-$n$. The unit of time is minutes, the unit of memory and final size on disk is gigabytes. Time and memory for our method is reported both when running fully in-memory, and when writing the final color sets to disk in 8 pieces (8 pcs). For all tools, the final disk includes only the coloring part of the data structure. The lowest time, memory and final disk values in each row are bolded.

| Dataset | Our method | | | | | Bifrost | | | GGCAT 2 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Time | Mem | Time (8 pcs) | Mem (8 pcs) | Final disk | Time | Mem | Final disk | Time | Mem | Final disk |
| S-128 | 0.22 | 0.47 | 0.40 | 0.41 | 0.00 | 0.20 | **0.24** | 0.01 | **0.11** | 0.69 | **0.00** |
| S-256 | 0.49 | 0.78 | 0.72 | 0.77 | 0.00 | 0.50 | **0.35** | 0.03 | **0.15** | 1.03 | **0.00** |
| S-512 | 1.10 | 1.46 | 1.61 | 1.42 | 0.04 | 1.49 | **0.58** | 0.09 | **0.23** | 1.50 | **0.01** |
| S-1024 | 2.32 | 1.70 | 3.33 | 1.67 | 0.11 | 3.52 | **0.87** | 0.21 | **0.39** | 2.43 | **0.03** |
| S-2048 | 5.30 | 2.64 | 7.58 | 2.38 | 0.33 | 8.83 | **1.56** | 0.51 | **0.76** | 4.10 | **0.09** |
| S-4096 | 10.88 | 2.93 | 15.51 | **2.51** | 0.86 | 22.57 | 3.06 | 1.27 | **1.44** | 6.37 | **0.24** |
| S-8192 | 21.48 | 4.58 | 31.06 | **3.37** | 2.13 | 57.45 | 5.73 | 2.76 | **2.48** | 12.33 | **0.52** |
| S-16384 | 50.10 | 11.26 | 76.82 | **6.38** | 6.46 | 172.74 | 16.32 | 8.28 | **6.09** | 22.41 | **2.18** |
| S-32768 | 131.68 | 24.13 | 202.63 | **10.13** | 16.45 | 435.19 | 34.88 | 15.12 | **13.69** | 29.69 | **10.53** |
| S-65536 | 286.90 | 47.59 | 437.45 | **13.74** | 39.71 | 1067.32 | 71.87 | 34.67 | **33.88** | 47.26 | **31.11** |
| R-128 | 0.50 | 3.57 | 1.07 | 3.52 | 0.00 | 0.74 | 1.64 | 0.10 | **0.14** | **1.41** | **0.00** |
| R-256 | 0.95 | 4.68 | 1.53 | 5.20 | 0.02 | 1.29 | 2.36 | 0.21 | **0.19** | **1.74** | **0.01** |
| R-512 | 2.02 | 7.13 | 3.25 | 7.72 | 0.07 | 2.68 | 4.03 | 0.50 | **0.30** | **2.49** | **0.03** |
| R-1024 | 4.27 | 7.36 | 6.54 | 7.66 | 0.21 | 5.73 | 6.78 | 1.02 | **0.55** | **3.64** | **0.08** |
| R-2048 | 8.48 | 9.06 | 12.57 | 8.86 | 0.62 | 11.96 | 10.82 | 1.73 | **0.95** | **4.98** | **0.21** |
| R-4096 | 17.33 | 11.52 | 25.38 | 10.56 | 1.72 | 26.19 | 18.47 | 3.79 | **1.68** | **7.04** | **0.55** |
| R-8192 | 36.76 | 17.47 | 53.19 | 14.42 | 4.58 | 60.17 | 33.25 | 8.55 | **3.15** | **11.50** | **1.38** |
| R-16384 | 80.90 | 29.57 | 114.33 | 21.46 | 12.41 | 148.18 | 60.28 | 17.65 | **6.04** | **19.07** | **3.61** |

**Table 2** Dataset statistics for $n \geq 128$ genomes. The Salmonella datasets are labeled S-$n$ and the random datasets are labeled R-$n$. The number of $k$-mers includes both DNA strands, so it is twice the number of canonical $k$-mers. The unitigs statistics are computed from the unidirected de Bruijn graph containing both DNA strands. The mean distinct set size is the average number of elements in the distinct color sets, and the mean color set size is the expected value of the size of the color set of a randomly chosen $k$-mer from the distinct $k$-mers.

| Dataset | Distinct $k$-mers | Key $k$-mers | Dense color sets | Sparse color sets | Mean distinct set size | Mean color set size | Number of unitigs | Mean unitig length |
|---|---|---|---|---|---|---|---|---|
| S-128 | 25,820,642 | 2.93 % | 2,593 | 15,822 | 57.96 | 47.68 | 737,677 | 65.00 |
| S-256 | 37,228,100 | 3.75 % | 14,059 | 111,781 | 141.79 | 66.57 | 1,372,922 | 57.12 |
| S-512 | 60,223,428 | 4.61 % | 130,209 | 557,266 | 268.83 | 82.04 | 2,734,306 | 52.03 |
| S-1024 | 79,681,694 | 4.88 % | 292,619 | 805,691 | 518.38 | 123.65 | 3,835,925 | 50.77 |
| S-2048 | 123,385,316 | 4.90 % | 535,700 | 1,207,623 | 964.60 | 160.11 | 5,925,366 | 50.82 |
| S-4096 | 170,807,566 | 5.21 % | 951,462 | 1,588,822 | 1743.32 | 229.74 | 8,686,367 | 49.66 |
| S-8192 | 219,025,616 | 5.31 % | 1,410,479 | 1,975,196 | 3217.66 | 357.85 | 11,241,222 | 49.48 |
| S-16384 | 355,596,480 | 5.76 % | 2,862,855 | 2,978,487 | 5512.34 | 440.83 | 19,737,346 | 48.02 |
| S-32768 | 514,359,252 | 5.70 % | 4,616,836 | 3,779,426 | 9367.01 | 609.37 | 28,289,327 | 48.18 |
| S-65536 | 679,565,754 | 5.80 % | 6,992,637 | 4,477,742 | 15911.33 | 921.01 | 37,961,602 | 47.90 |
| R-128 | 290,548,286 | 1.69 % | 50,749 | 137,831 | 15.63 | 3.64 | 4,876,243 | 89.58 |
| R-256 | 425,718,950 | 1.88 % | 330,717 | 375,599 | 26.62 | 4.83 | 7,918,143 | 83.76 |
| R-512 | 749,380,340 | 1.92 % | 894,982 | 667,385 | 47.60 | 5.50 | 14,239,936 | 82.63 |
| R-1024 | 1,266,299,900 | 1.98 % | 1,975,597 | 1,087,068 | 85.43 | 6.60 | 24,752,934 | 81.16 |
| R-2048 | 1,923,902,914 | 2.07 % | 3,874,407 | 1,540,177 | 136.79 | 8.63 | 39,455,816 | 78.76 |
| R-4096 | 3,016,705,654 | 2.24 % | 6,816,089 | 2,026,570 | 222.31 | 10.97 | 66,920,361 | 75.08 |
| R-8192 | 4,889,061,902 | 2.38 % | 11,929,411 | 2,627,387 | 337.43 | 13.57 | 114,873,088 | 72.56 |
| R-16384 | 7,577,410,586 | 2.59 % | 21,422,536 | 3,473,930 | 510.51 | 17.47 | 193,979,805 | 69.06 |

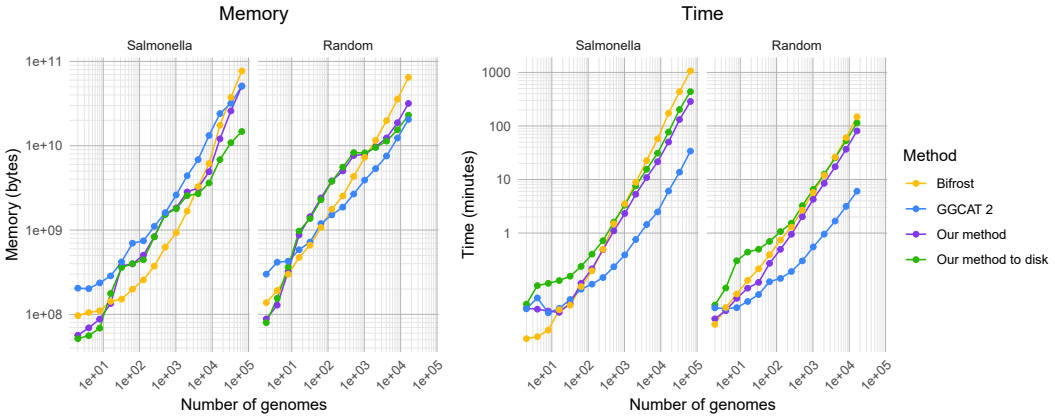



**Figure 6** Memory usage (in bytes) and runtime (in minutes) for various competing colored de Bruijn graph construction methods measured in our experiments. $k$ is set to 31. Note the double-log scale.

## 4.2 Datasets

We use the following two datasets:

- SALMONELLA: Randomly selected $2^{16} = 65536$ *Salmonella enterica* genomes from the AllTheBacteria [15] dataset.
- RANDOM: Randomly selected $2^{14} = 16384$ genomes from the AllTheBacteria [15] dataset.

We argue that these datasets represent the important performance extremes in the indexing problem at hand. The SALMONELLA dataset represents a low-diversity use case with large color sets, and RANDOM represents a high-diversity use case with small color sets. To study scaling of the methods, we run them on random subsets of $2, 4, 8, 16, \ldots$ genomes. Various key statistics on the datasets are listed in Table 2. All experiments were run with $k$-mer length $k = 31$ and fingerprint length $\ell = 128$.

## 4.3 Baselines

We compare our method against methods from the literature that are able to construct the color sets directly without first constructing an uncompressed representation. The competing methods are as follows:

- **GGCAT 2** [8] (commit `14e8159`). The construction algorithm in GGCAT 2 is a highly optimized minimizer bucketing method, which combines unitig construction and color indexing at the same time.
- **Bifrost** [14] (commit `b7659dd`). The algorithm first assembles the $k$-mers into a de Bruijn graph with minimizer hashing, and constructs the sets incrementally with dynamic data structures.

See Section 2.1 for a more detailed description of the inner workings of GGCAT 2 and Bifrost. We work under the assumption that the $k$-mers have already been indexed, so we do not include the time and space for $k$-mer indexing in our results, except for GGCAT 2, which merges the coloring and $k$-mer indexing phases in a way that they can not be separated from one another. In case of Bifrost, we modified the source code to add a timer that starts from the moment when the coloring process starts, and ends when the coloring is written to disk. In both full datasets SALMONELLA and RANDOM, the peak memory of Bifrost was observed to occur during color set construction, so no finer-grained memory measurement was needed.

We omit the pseudoalignment tools Fulgor [12] and Themisto [4] because their color construction algorithms are almost entirely outsourced to GGCAT. A recently published preprint introducing Cuttlefish 3 [19] claims a 3-4x speedup over GGCAT with comparable RAM using an external memory construction algorithm. As we are mostly interested in the RAM space, we leave Cuttlefish 3 out of the benchmark.

We also omit Metagraph [17] because the recommended way to construct the color sets is by first enumerating $k$-mer sets of colors (instead of color sets of $k$-mers), then transposing the relationship to obtain the color sets. This workflow blows up the intermediate space, because it is unable to compress the color information across genomes, where most of the redundancy lies, until later stages in the pipeline. Metagraph also provides a command-line option to construct directly in the final row-major form, but this does not appear to effectively exploit cross-genome redundancies until the end. For example, running annotation with options `--anno-type row`, `--fwd-and-reverse` and `--mem-cap 1gb` consumed 53 GiB of RAM already on 2048 Salmonella genomes, whereas our tool requires only 3 GiB. There are various other command line options and alternative workflows that could be employed, but designing such workflows requires substantial expertise from the user and knowledge of the structure of the input data, and showcasing a potentially sub-optimal workflow risks making Metagraph look worse than it is capable of, and thus we leave this out of scope of the present experimental study.

## 4.4 Results

Our method was run both fully in-memory, and in the mode where we construct the final sparse-dense structure directly to disk in *8 pieces* as described in Section 3.3. Time, memory and disk space of the final output for all benchmarked tools are listed in Table 1. Figure 6 shows time and memory plotted as the function of the number of genomes, and Figure 4 shows the scaling with respect to the number of threads.

### 4.4.1 Salmonella

In this dataset, the size of the color set structure dominates the size of the $k$-mer indexing structure. We find that GGCAT 2 is faster than both our approach and Bifrost by an order of magnitude. In absolute terms, the time to run our method is still reasonable: On the full dataset, Phases 1, 2 and 3 of our method took 20 min, 132 min and 132 min respectively when run in memory, and Phase 3 still takes only 280 min when run to disk. On the full Salmonella dataset, when the final index is constructed directly to disk, GGCAT uses more RAM than our method on every input size, and 3.4 times more on the full dataset. Despite the three tools using significantly different color set representations, the final disk sizes are similar: 40GB for our sparse-dense index and the SBWT-based $k$-mer index, 35GB for Bifrost and 31GB for GGCAT 2. The construction space overhead, that is, $m/d - 1$, where $m$ is the peak memory and $d$ is final disk size, is 20% for our fully in-memory method, 242% for Bifrost and 52% for GGCAT 2. When the final index is constructed directly to disk, our overall peak RAM is only roughly one third of the final index size.

The mean unitig length decreases as the number of unitigs grows, stabilizing at around 48 base pairs (see Table 2). Since the last $k$-mer of every unitig is always marked as a key $k$-mer, we have to mark at least $1/(48 - k + 1) \approx 5.6\%$ of the $k$-mers as key. We end up marking only a slightly higher amount: a total of 5.8% of all $k$-mers. Overall, the fraction of key $k$-mers seems to be quite well approximated by the fraction of unitig-ending $k$-mers $1/(L - k + 1)$, where $L$ is the mean unitig length (see Figure 5).

For a small number of genomes, a vast majority of the distinct color sets are sparse, but at 32,768 genomes, the number of dense sets overtakes the number of sparse sets. In the full dataset, the average distinct color set contains almost one quarter of all colors, making the bitmap representation highly efficient.

### 4.4.2 Random

In this dataset, the color set structure is small, and the size of the $k$-mer index dominates. The average unitig length was only 69 base pairs on the full dataset. This is consistent with the low number of key $k$-mers, which accounted for only 2.59% of the total $k$-mers, compared to 5.80% in the largest Salmonella dataset. However, this did not translate into reduced space compared to the Salmonella subsets with the same number of genomes, because, even though the sparse-dense representation of the distinct color sets is smaller, the space peak during fingerprinting is much higher.

GGCAT 2 was the best in all the three metrics of time, memory and final disk space. However, in terms of space, our method comes a close second, consuming 21 GiB RAM when the final index is constructed directly to disk, whereas GGCAT takes 19 GiB. For our method, phases 1, 2 and 3 took 11 min, 36 min and 33 min respectively. When writing the final index directly to disk, the time of phase 3 increases to 67 min. The trends in Figure 6 suggest that if one more data point was added, our method might use less space than GGCAT 2.

## 5 Concluding Remarks

Given the exploratory nature of genomics, indexing is a frequent operation, making more efficient index construction algorithms, such as the one we have described, of continuing interest.

One direction for future work is to make use of better perfect hashing within our construction algorithm. The current state-of-the-art in minimal perfect hashing, PtrHash [13], claims about 21ns to hash an integer key and 2.41 bits per key. This could lead to significant improvements in our context. Another direction concerns running the construction process from unitigs, which would remove the need for the within-color deduplication structure and so might decrease the peak space and/or improve running time. Preprocessing to unitigs is expensive, however, if the data is already in this form, e.g., unitigs from the Logan project [7], this approach may prove interesting.

Finally, beyond index construction, our approach allows efficient $n$-way merging of colored representations, which is of interest for index updates and may allow more efficient $k$-mer set operations in the context of genomic data exploration [10].

## References


1 Jarno N Alanko, Elena Biagi, and Simon J Puglisi. Longest common prefix arrays for succinct k-spectra. In *International Symposium on String Processing and Information Retrieval*, pages 1–13. Springer, 2023. doi:10.1007/978-3-031-43980-3_1.

2 Jarno N Alanko, Elena Biagi, and Simon J Puglisi. Finimizers: Variable-length bounded-frequency minimizers for $k$-mer sets. *IEEE Transactions on Computational Biology and Bioinformatics*, 2025.

3 Jarno N. Alanko, Simon J. Puglisi, and Jaakko Vuohtoniemi. Small searchable k-spectra via subset rank queries on the spectral Burrows-Wheeler transform. In *Proc. ACDA*, pages 225–236. SIAM, 2023. doi:10.1137/1.9781611977714.20.

4 Jarno N Alanko, Jaakko Vuohtoniemi, Tommi Mäklin, and Simon J Puglisi. Themisto: a scalable colored k-mer index for sensitive pseudoalignment against hundreds of thousands of bacterial genomes. *Bioinformatics*, 39(Supplement_1):i260–i269, 2023.

5 Fatemeh Almodaresi, Prashant Pandey, and Rob Patro. Rainbowfish: a succinct colored de bruijn graph representation. *bioRxiv*, page 138016, 2017.

6 Alessio Campanelli, Giulio Ermanno Pibiri, and Rob Patro. Fast pseudoalignment queries on compressed colored de bruijn graphs. In *Proc. 25th International Conference on Algorithms for Bioinformatics (WABI)*, volume 344 of *LIPIcs*, pages 6:1–6:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.WABI.2025.6.

7 Rayan Chikhi, Brice Raffestin, Anton Korobeynikov, Robert Edgar, and Artem Babaian. Logan: Planetary-scale genome assembly surveys life's diversity. *bioRxiv*, 2024. doi:10.1101/2024.07.30.605881.

8 Andrea Cracco and Alexandru I Tomescu. Extremely fast construction and querying of compacted and colored de bruijn graphs with ggcat. *Genome Research*, 33(7):1198–1207, 2023.

9 Daniel Danciu, Mikhail Karasikov, Harun Mustafa, André Kahles, and Gunnar Rätsch. Topology-based sparsification of graph annotations. *Bioinformatics*, 37(Supplement_1):i169–i176, 2021.

10 Bastien Degardins, Charles Paperman, and Camille Marchet. Vizitig: a pangenome and pantranscriptome explorer. *bioRxiv*, 2026. doi:10.1101/2025.04.19.649656.

11 Emmanuel Esposito, Thomas Mueller Graf, and Sebastiano Vigna. Recsplit: Minimal perfect hashing via recursive splitting. In *2020 Proceedings of the Twenty-Second Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 175–185. SIAM, 2020. doi:10.1137/1.9781611976007.14.

12 Jason Fan, Jamshed Khan, Noor Pratap Singh, Giulio Ermanno Pibiri, and Rob Patro. Fulgor: a fast and compact k-mer index for large-scale matching and color queries. *Algorithms Mol. Biol.*, 19(1):3, 2024. doi:10.1186/S13015-024-00251-9.

13 Ragnar Groot Koerkamp. Ptrhash: Minimal perfect hashing at ram throughput. In *23rd International Symposium on Experimental Algorithms (SEA 2025)*, pages 21:1–21:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.SEA.2025.21.

14 Guillaume Holley and Páll Melsted. Bifrost: highly parallel construction and indexing of colored and compacted de Bruijn graphs. *Genome biology*, 21(1):1–20, 2020.

15 Martin Hunt, Leandro Lima, Daniel Anderson, George Bouras, Michael Hall, Jane Hawkey, Oliver Schwengers, Wei Shen, John A. Lees, and Zamin Iqbal. Allthebacteria – all bacterial genomes assembled, available, and searchable. *bioRxiv*, 2025. doi:10.1101/2024.03.08.584059.

16 Zamin Iqbal, Mario Caccamo, Isaac Turner, Paul Flicek, and Gil McVean. De novo assembly and genotyping of variants using colored de Bruijn graphs. *Nature genetics*, 44(2):226–232, 2012.

17 Mikhail Karasikov, Harun Mustafa, Daniel Danciu, Oleksandr Kulkov, Marc Zimmermann, Christopher Barber, Gunnar Rätsch, and André Kahles. Efficient and accurate search in petabase-scale sequence repositories. *Nature*, pages 1–9, 2025.

18 Mikhail Karasikov, Harun Mustafa, Amir Joudaki, Sara Javadzadeh-No, Gunnar Rätsch, and André Kahles. Sparse binary relation representations for genome graph annotation. *Journal of Computational Biology*, 27(4):626–639, 2020. doi:10.1089/CMB.2019.0324.

19 Jamshed Khan, Laxman Dhulipala, and Rob Patro. Fast and scalable parallel external-memory construction of colored compacted de bruijn graphs with cuttlefish 3. *bioRxiv*, 2025.

20 Martin D Muggli, Alexander Bowe, Noelle R Noyes, Paul S Morley, Keith E Belk, Robert Raymond, Travis Gagie, Simon J Puglisi, and Christina Boucher. Succinct colored de Bruijn graphs. *Bioinformatics*, 33(20):3181–3187, October 2017. doi:10.1093/bioinformatics/btx067.

21 Gonzalo Navarro. *Compact data structures: A practical approach*. Cambridge University Press, 2016.

**22** G. E. Pibiri. Sparse and skew hashing of K-mers. *Bioinformatics*, 38(Suppl 1):i185–i194, June 2022. doi:10.1093/BIOINFORMATICS/BTAC245.

**23** Giulio Ermanno Pibiri and Rossano Venturini. Techniques for inverted index compression. *ACM Comput. Surv.*, 53(6):125:1–125:36, 2021. doi:10.1145/3415148.

**24** Albert L. Zobrist. A new hashing method with application for game playing. Technical Report TR88, University of Wisconsin–Madison Department of Computer Sciences, 1970. URL: https://minds.wisconsin.edu/handle/1793/57624.

# Theoretically and Practically Faster Algorithms for Protein Structure Alignment

**Masahito Tsukahara**
Human Genome Center, The Institute of Medical Science, The University of Tokyo, Japan
Department of Computer Science, Graduate School of Information Science and Technology, The University of Tokyo, Japan

**Tetsuo Shibuya**
Human Genome Center, The Institute of Medical Science, The University of Tokyo, Japan

## Abstract

Identifying shared substructures in 3D protein models is essential for structural bioinformatics. This task can be modeled as a sequential Largest Common Point-set (LCP) problem under the bottleneck distance. We propose a new $O(n^{13} \log n)$-time exact algorithm for this problem, which improves upon the previous best-known complexity of $O(n^{14})$, where $n$ is the maximum size of the two input structures. Since these theoretical bounds are practically too large, an $O(n^7 \log n)$-time approximation algorithm with solution-size guarantees has been proposed; however, it remains too time-consuming for practical applications. Thus, we also propose a new filtering technique to enhance the approximation algorithm without increasing the theoretical time complexity or losing the solution-size guarantees. Experiments with PDB data show that our technique achieves over a 24-fold speedup at $n = 130$. While the previous algorithm required 3.80 hours on average for $n = 130$ in our experiments, making it difficult to test larger structures, our algorithm can process $n = 200$ in only 2.33 hours on average.

**2012 ACM Subject Classification** Theory of computation → Approximation algorithms analysis; Applied computing → Molecular structural biology

**Keywords and phrases** Structural Bioinformatics, Pairwise Alignment, Exact Algorithm, Approximation Algorithm, Computational Geometry

**Digital Object Identifier** 10.4230/LIPIcs.WABI.2026.7

**Supplementary Material**
*Software (Source Code)*: `https://github.com/masat03110/ProteinStructureAlignmentPlus` [18] archived at `swh:1:dir:8bdeadc1f4304d35619d3600f97069c17bc38192`

**Funding** This work was partially supported by KAKENHI Grant Numbers 23K28035.

## 1 Introduction

Identifying shared 3D substructures among protein molecules is a fundamental task in structural bioinformatics. This task is commonly known as protein structure alignment, and it provides important clues about protein function, molecular interactions, and structure prediction [7, 15]. Several measures have been used to evaluate protein structure alignments, including Root Mean Square Deviation (RMSD) [3], the Global Distance Test (GDT) score, Local/Global Alignment (LGA) [24], and the TM-score [25]. Among them, GDT has been widely used in the Critical Assessment of protein Structure Prediction (CASP) experiments [16], because it is less sensitive than RMSD to local deviations and outliers [23]. Its relevance continues in recent CASP evaluations, including CASP16 [22].

Motivated by the GDT score, we study protein structure alignment through the Largest Common Point-set (LCP) problem under the bottleneck distance. Given two point sets $P$ and $Q$ in $\mathbb{R}^3$ and a threshold $\epsilon > 0$, the LCP problem asks for a maximum-cardinality subset $S \subseteq P$ such that, after a rigid transformation $T$, every point in $T(S)$ is within distance $\epsilon$ of


© Masahito Tsukahara and Tetsuo Shibuya;
licensed under Creative Commons License CC-BY 4.0

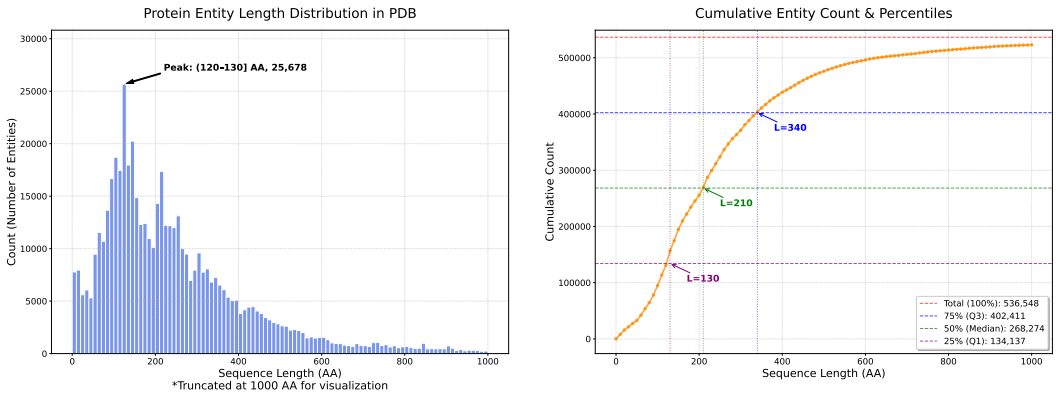



**Figure 1** Distribution of protein sequence lengths in the Protein Data Bank (PDB) [6] as of May 9, 2026. **(Left)** Histogram of sequence lengths, truncated at 1,000 amino acids for visualization; the highest bin is (120, 130] residues. **(Right)** Cumulative count of protein entities ($N = 536{,}548$). The 25%, 50%, and 75% percentiles correspond to $L = 130$, $L = 210$, and $L = 340$, respectively.

a distinct point in $Q$ [10]. For proteins, however, residues are not merely unordered points: their sequence order along the backbone is biologically essential. We therefore focus on the sequential LCP (sLCP) problem, which preserves residue order and naturally formulates the protein structure alignment problem considered in this paper.

A central difficulty is to obtain algorithms that are both theoretically guaranteed and practically scalable. The exact algorithm of Ambühl et al. [2] was previously reported to run in $O(n^{32})$ time, where $n$ is the maximum size of the two input structures; however, the stated running-time bound does not necessarily follow from the reported analysis.[1] Although adapting Choi and Goyal's framework [9] gives an $O(n^{14})$-time exact algorithm, it remains impractical. Heuristic methods have also been developed [11, 20], but lack rigorous solution-quality guarantees. Approximation algorithms with theoretical guarantees have been developed to bridge this gap. Akutsu [1] gave an $O(n^8)$-time algorithm, followed by algorithms with stronger guarantees and better efficiency [8, 17]. Li and Ng [14] proposed an important framework with a reported running time of $O(\Delta_\epsilon^2 n^7/\delta^5)$, where $\Delta_\epsilon$ and $\delta$ are defined later, but Tsukahara and Shibuya [19] showed that this bound does not hold in the worst case and proposed faster, more accurate corrected algorithms running in $O(\Delta_\epsilon n^7 \log n/\delta^5)$ time. Nevertheless, even their fastest algorithm remains costly in our experiments, requiring 1.54 hours at $n = 120$ and 3.80 hours at $n = 130$ on average. This matters for PDB-scale applications [6]: as shown in Figure 1, PDB entity lengths peak in (120, 130] residues, so under our experimental time limit, the previous fastest algorithms are practically limited to about the shortest 25% of PDB entities and become costly near the most common size range.

In this paper, we address both the theoretical and practical sides of this problem. First, we adapt and improve the framework of Choi and Goyal [9], obtaining an exact algorithm running in $O(\Delta_\epsilon n^{13} \log n)$ time. We then introduce a filtering technique for the approximation framework of Li and Ng [14] and Tsukahara and Shibuya [19], preserving its theoretical guarantees while improving practical performance. Experiments on PDB data show comparable solution quality for $n = 10, \ldots, 50$ and much better scalability: our method finishes in 5.83 and 9.50 minutes at $n = 120$ and $n = 130$, respectively, achieves over a 24-fold speedup at $n = 130$, and processes $n = 200$ and $n = 210$ in 2.33 and 3.33 hours, respectively, covering nearly 50% of PDB entities.

[1] This issue was pointed out via personal communication in May 2026, with acknowledgment to Bernd Gärtner for his insights.

## 2 Preliminaries

In this section, we introduce the notation used throughout the paper and formally define the Protein Structure Alignment problem. We model a protein as an ordered sequence of 3D points, where each point represents a residue, namely its $C\alpha$ atom. Thus, a protein $\mathcal{P}$ of length $n$ is written as $\mathcal{P} = (p_1, \ldots, p_n)$, where $p_i \in \mathbb{R}^3$ is the position of the $i$-th residue. We use the following standard bounded-density property of protein structures.

▶ **Property 1** (Bounded Density [21]). *Due to steric constraints, residues in a protein cannot be arbitrarily close to each other. If the minimum distance between any two residues is at least $D_1$, then the number of residues contained in any ball of radius $\epsilon$ is bounded by $\Delta_\epsilon = O((1 + \frac{2\epsilon}{D_1})^3)$. In particular, for fixed $\epsilon$, $\Delta_\epsilon$ is independent of the protein length.*

A *rigid transformation* is a mapping $T : \mathbb{R}^3 \to \mathbb{R}^3$ consisting of a rotation and a translation. It preserves Euclidean distances, that is, $D(p, q) = D(T(p), T(q))$ for all $p, q \in \mathbb{R}^3$, where $D$ denotes the Euclidean distance. For a sequence $\mathcal{P} = (p_1, \ldots, p_n)$, we write $T(\mathcal{P}) = (T(p_1), \ldots, T(p_n))$.

Let $\mathcal{S} = (s_1, \ldots, s_k)$ and $\mathcal{S}' = (s'_1, \ldots, s'_k)$ be two point sequences of the same length. A bijection $f : \mathcal{S} \to \mathcal{S}'$ is an order-preserving one-to-one correspondence defined by $f(s_i) = s'_i$ for each $i$. We say that $\mathcal{S}$ is $\epsilon$-*congruent* to $\mathcal{S}'$ if there exists a rigid transformation $T$ such that $D(T(\mathcal{S}), \mathcal{S}') \coloneqq \max_{1 \le i \le k} D(T(s_i), s'_i) \le \epsilon$.

For two proteins $\mathcal{P}$ and $\mathcal{Q}$, we seek the longest subsequence of $\mathcal{P}$ that is $\epsilon$-congruent to a subsequence of $\mathcal{Q}$. Let $\mathfrak{S}$ be the set of all subsequences of $\mathcal{P}$ that are $\epsilon$-congruent to some subsequence of $\mathcal{Q}$. We denote an optimal subsequence by $sLCP(\mathcal{P}, \mathcal{Q}, \epsilon) \in \arg\max_{\mathcal{S} \in \mathfrak{S}} |\mathcal{S}|$. Based on these definitions, the main problem considered in this paper is formulated as follows.

Protein Structure Alignment Problem

**Input:** Two proteins $\mathcal{P}$ and $\mathcal{Q}$, and a distance threshold $\epsilon > 0$.

**Output:** A subsequence $\mathcal{S}$ of $\mathcal{P}$, a rigid transformation $T$, and a bijection $f$ from $\mathcal{S}$ to a subsequence of $\mathcal{Q}$ such that $|\mathcal{S}| = |sLCP(\mathcal{P}, \mathcal{Q}, \epsilon)|$ and $D(T(\mathcal{S}), f(\mathcal{S})) \le \epsilon$.

Unless otherwise stated, we let $n$ and $m$ denote the lengths of $\mathcal{P}$ and $\mathcal{Q}$, respectively, and assume $n \ge m$ without loss of generality. We consider the nontrivial case $|sLCP(\mathcal{P}, \mathcal{Q}, \epsilon)| \ge 2$, since instances with optimum size at most one can be solved immediately. For the time-complexity analysis, we use the real RAM model of computation, in which basic arithmetic operations on real numbers are exact and take constant time.

## 3 Exact Algorithms

In this section, we discuss exact algorithms for the Protein Structure Alignment problem. As noted in the introduction, the exact algorithm of Ambühl et al. [2], previously reported to run in $O(n^{32})$ time, was once regarded as the fastest exact method for this problem. However, the stated running-time bound does not necessarily follow from the reported analysis. Thus, a reliable theoretical baseline with a provable polynomial-time bound is needed.

We show that the algorithm of Choi and Goyal [9], originally developed for point pattern matching, can be adapted to the sequential setting to solve the problem in $O(n^{14})$ time. We first describe this adaptation and then improve it by exploiting the Bounded Density property (Property 1), obtaining an exact algorithm with running time $O(\Delta_\epsilon n^{13} \log n)$.

## 3.1 A Baseline Exact Algorithm: Adapting Choi and Goyal's Framework

Let $\mathfrak{T}$ denote the space of all rigid motions in $\mathbb{R}^3$. We identify $\mathfrak{T}$ with $\mathbb{R}^3 \times SO(3)$, corresponding to translations and rotations, respectively. For two points $p, q \in \mathbb{R}^3$ and a threshold $\epsilon > 0$, define

$$\text{score}(p, q, \epsilon) = \begin{cases} 1 & \text{if } \|p - q\| \leq \epsilon, \\ 0 & \text{otherwise.} \end{cases}$$

Given two proteins $\mathcal{P} = (p_1, \ldots, p_n)$ and $\mathcal{Q} = (q_1, \ldots, q_m)$, two rigid motions $T_1, T_2 \in \mathfrak{T}$ are said to be equivalent if $\text{score}(T_1(p_i), q_j, \epsilon) = \text{score}(T_2(p_i), q_j, \epsilon)$ for all $1 \leq i \leq n$ and $1 \leq j \leq m$. The induced partition of $\mathfrak{T}$ is denoted by $\Pi(\mathcal{P}, \mathcal{Q}, \epsilon)$, and each equivalence class is called a cell. Although such a cell need not be connected, it is characterized by an $n \times m$ binary score table whose $(i, j)$-entry is $\text{score}(T(p_i), q_j, \epsilon)$ for any $T$ in the cell.

We use the following theorem of Choi and Goyal [9].

▶ **Theorem 2** ([9]). *For any cell $F \in \Pi(\mathcal{P}, \mathcal{Q}, \epsilon)$, there exists a set $C$ of $k \leq 6$ contact constraints whose satisfying rigid motions form $2^{O(1)}$ connected components, at least one of which lies in the closure of $F$.*

This theorem implies that it suffices to enumerate all sets of at most six contact constraints and examine representative rigid motions obtained from them. Here, a contact constraint corresponds to a pair $(p_i, q_j)$ whose distance becomes exactly $\epsilon$ after applying the rigid motion to $p_i$. The resulting adaptation gives the following algorithm.

**Step 1.** **Enumerate constraints:** Each contact constraint corresponds to a pair $(p_i, q_j) \in \mathcal{P} \times \mathcal{Q}$ for which $T(p_i)$ lies on the boundary of the $\epsilon$-ball centered at $q_j$. We enumerate all combinations of $k \leq 6$ such constraints, giving $O(n^6 m^6)$ combinations.

**Step 2.** **Compute transformations:** For each combination, compute representative rigid motions satisfying the corresponding constraints. Since the dimension is fixed to three and the number of constraints is at most six, this can be done in constant time under the real RAM model, for example by using the method of Basu et al. [4].

**Step 3.** **Dynamic programming:** For each representative rigid motion $T$, construct the $n \times m$ score table in $O(nm)$ time. We then compute the maximum number of matched residue pairs that preserve the sequence order under this fixed transformation, which we call the sequential alignment size for $T$. This value is obtained by the dynamic programming recurrence of Akutsu [1]: $A(i, j) = \max\{A(i-1, j), A(i, j-1), A(i-1, j-1) + \text{score}(T(p_i), q_j, \epsilon)\}$, where $A(i, j) = 0$ if $i = 0$ or $j = 0$.

**Step 4.** **Output recovery:** Choose the rigid motion maximizing $A(n, m)$ and recover the optimal subsequence $\mathcal{S}$ and bijection $f$ by standard DP backtracking.

For each of the $O(n^6 m^6)$ combinations, the dynamic programming step takes $O(nm)$ time. Thus, since $n \geq m$, we obtain the following theorem.

▶ **Theorem 3.** *The Protein Structure Alignment problem is solvable in $O(n^{14})$ time.*

## 3.2 An Improved Exact Algorithm

We now improve the baseline exact algorithm by avoiding the dense $n \times m$ score table for each candidate transformation. For a fixed rigid transformation $T$, define the active pair set as $E_T = \{(i, j) \mid D(T(p_i), q_j) \leq \epsilon\}$. Only pairs in $E_T$ have score one and can contribute to the dynamic programming value; all other entries of the score table are zero. The baseline algorithm constructs the full score table, whereas our algorithm enumerates only $E_T$.

**Algorithm 1** Improved Exact Algorithm for Protein Structure Alignment.

**Input** : Two proteins $\mathcal{P}$ ($|\mathcal{P}| = n$) and $\mathcal{Q}$ ($|\mathcal{Q}| = m$), and a distance threshold $\epsilon > 0$
**Output** : A subsequence $\mathcal{S}$ of $\mathcal{P}$, a rigid transformation $T$, and a bijection $f$

$T \leftarrow$ identity transformation, $s_{\max} \leftarrow 0, E_{\max} \leftarrow \emptyset$ ;
$\mathfrak{T} \leftarrow$ set of candidate transformations ; `// Steps 1 and 2 in Sec. 3.1`
$H \leftarrow$ map grouping each $q_j \in \mathcal{Q}$ by grid key $(\lfloor x_{q_j}/\epsilon \rfloor, \lfloor y_{q_j}/\epsilon \rfloor, \lfloor z_{q_j}/\epsilon \rfloor)$ ;
**for** $T_c \in \mathfrak{T}$ **do**
  $E_{T_c} \leftarrow \emptyset$ ;
  **for** $i = 1$ **to** $n$ **do**
    $p'_i \leftarrow T_c(p_i)$ ;
    **for** each $q_j \in$ GetFrom27NeighborCells$(H, p'_i, \epsilon)$ **do**
      **if** $D(p'_i, q_j) \leq \epsilon$ **then**
        $E_{T_c} \leftarrow E_{T_c} \cup \{(i, j)\}$ ; `// Active pair`
  $s_c \leftarrow$ CalculateAlignmentScore$(E_{T_c})$ ; `// Sparse DP in [19]`
  **if** $s_c > s_{\max}$ **then**
    $s_{\max} \leftarrow s_c, T \leftarrow T_c, E_{\max} \leftarrow E_{T_c}$ ;
Recover $\mathcal{S}$ and $f$ from $E_{\max}$ by sparse DP backtracking ; `// Step 4 in Sec. 3.1`
**return** $\mathcal{S}$, $T$, $f$ ;

Our improvement is based on the Bounded Density property (Property 1). Since $T$ preserves inter-residue distances, any ball of radius $\epsilon$ contains at most $O(\Delta_\epsilon)$ points of $T(\mathcal{P})$. Therefore, over all points of $\mathcal{Q}$, the number of active pairs is $|E_T| = O(\Delta_\epsilon m)$. Algorithm 1 exploits this sparsity using two procedures, which we describe below.

First, GetFrom27NeighborCells is a spatial lookup procedure for retrieving candidates near a transformed point. We partition $\mathcal{Q}$ into grid cells of side length $\epsilon$ and store each point $q_j$ in a map $H$ indexed by its grid key. Given a point $p' = T(p_i)$, GetFrom27NeighborCells$(H, p', \epsilon)$ returns all points of $\mathcal{Q}$ stored in the cell containing $p'$ and its 26 neighboring cells. Any point within distance $\epsilon$ from $p'$ must lie in one of these 27 cells, so checking these candidates and applying the distance test exactly recovers all pairs $(i, j) \in E_T$. Using a balanced map such as a B-tree [5], $H$ is constructed in $O(m \log m)$ time, and all active pairs for a fixed $T$ are retrieved in $O(\Delta_\epsilon n \log m + |E_T|)$ time because each cell contains $O(\Delta_\epsilon)$ points. Since $|E_T| = O(\Delta_\epsilon m)$ and $n \geq m$, this is bounded by $O(\Delta_\epsilon n \log m)$.

Second, CalculateAlignmentScore is the sparse dynamic programming procedure applied to the active pair set. Given $E_T$, it computes the maximum number of active pairs that can be selected while preserving the order of both sequences; equivalently, it computes the sequential alignment size for the fixed transformation $T$. We implement this procedure using the sparse DP framework of Tsukahara and Shibuya [19], which runs in $O(|E_T| \log m)$ time. Since $|E_T| = O(\Delta_\epsilon m)$ and $n \geq m$, this is bounded by $O(\Delta_\epsilon n \log m)$.

Thus, for each candidate transformation, Algorithm 1 replaces the dense $O(nm)$ score-table construction and DP with sparse pair retrieval and sparse DP in $O(\Delta_\epsilon n \log m)$ time, up to lower-order terms. Applying this procedure to all $O(n^6 m^6)$ candidate transformations and using $n \geq m$, we obtain the following theorem.

▶ **Theorem 4.** *The Protein Structure Alignment problem is solvable in $O(\Delta_\epsilon n^{13} \log n)$ time.*

# 4 Approximation Algorithms

The exact algorithm in the previous section, even with the improved $O(\Delta_\epsilon n^{13} \log n)$ running time, is still computationally prohibitive for large protein structures. To obtain theoretically guaranteed solutions more efficiently, Tsukahara and Shibuya [19] corrected and extended the approximation framework of Li and Ng [14]. Their result gives the following guarantee.

▶ **Theorem 5** ([19]). *Given two proteins $\mathcal{P}$ and $\mathcal{Q}$, a distance threshold $\epsilon > 0$, and an approximation factor $0 < \delta \leq 3$, there exists an algorithm that computes a subsequence $\hat{\mathcal{S}}$ of $\mathcal{P}$, a rigid transformation $\hat{T}$, and a bijection $\hat{f}$ from $\hat{\mathcal{S}}$ to a subsequence of $\mathcal{Q}$ such that $|\hat{\mathcal{S}}| \geq |sLCP(\mathcal{P}, \mathcal{Q}, \epsilon)|$ and $D(\hat{T}(\hat{\mathcal{S}}), \hat{f}(\hat{\mathcal{S}})) \leq (1+\delta)\epsilon$ in $O(\Delta_\epsilon n^7 \log n/\delta^5)$ time, where $n = \max(|\mathcal{P}|, |\mathcal{Q}|)$.*

Although this is much faster than exact algorithms, its practical scalability is still limited. As shown in Section 5, even the fastest existing guaranteed algorithm becomes costly around $n = 120$ to 130, which is near the most common size range in the PDB. We therefore propose an improved approximation algorithm based on a pruning technique. Our method preserves the guarantee of Theorem 5 while substantially improving practical efficiency. We first review the radial-pair framework underlying Li and Ng [14] and Tsukahara and Shibuya [19], and then present our improvement.

## 4.1 The Approximation Framework

The framework relies on the notion of a *radial pair*. For a point sequence $\mathcal{P}$, a point $p' \in \mathcal{P}$ is a radial point of $p \in \mathcal{P}$ if it is farthest from $p$ among all points in $\mathcal{P}$. The pair $(p, p')$ is called a radial pair of $\mathcal{P}$. Note that $(p', p)$ is not necessarily a radial pair of $\mathcal{P}$. The following lemma of Li et al. [12] is the geometric basis of the approximation framework.

▶ **Lemma 6** ([12]). *Let $\mathcal{P}$ be a point sequence, $T_1$ and $T_2$ be rigid transformations, and $(p, p')$ be a radial pair of $\mathcal{P}$. If $D(T_1(p), T_2(p)) \leq \gamma$ and $D(T_1(p'), T_2(p')) \leq \gamma$, then there exists a rotation $R$ around the axis, namely the line through $T_1(p)$ and $T_1(p')$, such that $D(R(T_1(\tilde{p})), T_2(\tilde{p})) \leq 3\gamma$ for all $\tilde{p} \in \mathcal{P}$.*

Using this geometric property, Tsukahara and Shibuya [19] obtained the following subroutine for the Protein Structure Alignment problem.

▶ **Lemma 7** ([19]). *Given two proteins $\mathcal{P}$ and $\mathcal{Q}$, a distance threshold $\epsilon > 0$, an approximation factor $0 < \delta \leq 3$, a radial pair $(p, p')$ of an optimal $sLCP(\mathcal{P}, \mathcal{Q}, \epsilon)$, and a pair $(q, q')$ in $\mathcal{Q}$ such that $f(p) = q$ and $f(p') = q'$ for an optimal bijection $f$, there exists an $O(\Delta_\epsilon n^3 \log n/\delta^5)$-time algorithm that computes a subsequence $\hat{\mathcal{S}}$ of $\mathcal{P}$, a rigid transformation $\hat{T}$, and a bijection $\hat{f}$ satisfying the guarantee of Theorem 5, where $n = \max(|\mathcal{P}|, |\mathcal{Q}|)$.*

**Proof.** Let $\mathcal{S}^* = sLCP(\mathcal{P}, \mathcal{Q}, \epsilon)$, and let $T^*$ and $f^*$ be an optimal rigid transformation and bijection for $\mathcal{S}^*$. Thus, $D(T^*(\mathcal{S}^*), f^*(\mathcal{S}^*)) \leq \epsilon$. By assumption, $(p, p')$ is a radial pair of $\mathcal{S}^*$ and $f^*(p) = q$, $f^*(p') = q'$.

The algorithm of Tsukahara and Shibuya [19], based on the sampling framework of Li and Ng [14], enumerates $O(1/\delta^5)$ candidate transformations determined by the fixed pairs $(p, p')$ and $(q, q')$. Among these candidates, there is a transformation $T'$ such that the images of the two radial points are sufficiently close to their positions under the optimal alignment. By Lemma 6, this implies that there exists a rotation $R$ around the axis $l(T'(p), T'(p'))$ such that, for every $\tilde{p} \in \mathcal{S}^*$, the transformed point $R(T'(\tilde{p}))$ is within distance $\delta\epsilon$ of $T^*(\tilde{p})$. Therefore, for every $\tilde{p} \in \mathcal{S}^*$, $D(R(T'(\tilde{p})), f^*(\tilde{p})) \leq D(R(T'(\tilde{p})), T^*(\tilde{p})) + D(T^*(\tilde{p}), f^*(\tilde{p})) \leq \delta\epsilon + \epsilon = (1+\delta)\epsilon$.

**Algorithm 2** Baseline Approximation Algorithm for Protein Structure Alignment [19].

**Input** : Two proteins $\mathcal{P}$ ($|\mathcal{P}| = n$) and $\mathcal{Q}$ ($|\mathcal{Q}| = m$), a distance threshold $\epsilon > 0$, and an approximation factor $0 < \delta \leq 3$

**Output** : A subsequence $\hat{\mathcal{S}}$ of $\mathcal{P}$, a rigid transformation $\hat{T}$, and a bijection $\hat{f}$

1 $\hat{\mathcal{S}} \leftarrow (), \hat{T} \leftarrow$ identity transformation, $\hat{f} \leftarrow$ empty bijection;
2 **for** $(p_{i_1}, p_{i_2}) \in \mathcal{P} \times \mathcal{P}$ *with* $i_1 \neq i_2$ **do**
3  **for** $(q_{j_1}, q_{j_2}) \in \mathcal{Q} \times \mathcal{Q}$ *with* $j_1 \neq j_2$ **do**
4  Calculate $\mathcal{S}_c, T_c, f_c$ from $(\mathcal{P}, \mathcal{Q}, \epsilon, \delta, (p_{i_1}, p_{i_2}), (q_{j_1}, q_{j_2}))$ ; `// Lemma 7`
5  **if** $|\mathcal{S}_c| > |\hat{\mathcal{S}}|$ **then**
6  $\hat{\mathcal{S}} \leftarrow \mathcal{S}_c, \hat{T} \leftarrow T_c, \hat{f} \leftarrow f_c$;
7 **return** $\hat{\mathcal{S}}$, $\hat{T}$, $\hat{f}$;

Hence, during the enumeration, the algorithm necessarily considers a transformation under which all points of the optimal subsequence $\mathcal{S}^*$ can be matched to $\mathcal{Q}$ within distance $(1+\delta)\epsilon$ while preserving the sequence order. The dynamic programming step for that transformation therefore returns a subsequence of size at least $|\mathcal{S}^*|$.

For each of the $O(1/\delta^5)$ sampled candidates, the best order-preserving matching can be computed in $O(\Delta_\epsilon n^3 \log n)$ time by the sparse dynamic programming procedure of Tsukahara and Shibuya [19]. Thus the total running time is $O(\Delta_\epsilon n^3 \log n/\delta^5)$. ◀

Lemma 7 gives the desired approximation once a radial pair of the optimal solution and its image pair in $\mathcal{Q}$ are fixed. Since these pairs are unknown, the baseline framework enumerates all $O(n^2)$ candidate pairs in $\mathcal{P}$ and all $O(m^2)$ image pairs in $\mathcal{Q}$, applies the fixed-pair procedure of Lemma 7 to each quadruple, and returns the largest solution. This procedure is summarized in Algorithm 2.

The enumeration considers $O(n^2m^2)$ quadruples, each requiring $O(\Delta_\epsilon n^3 \log n/\delta^5)$ time. Thus, for $n \geq m$, the total running time is $O(\Delta_\epsilon n^7 \log n/\delta^5)$. Since one quadruple matches the optimal radial pair and image pair, Theorem 5 follows.

## 4.2 An Improved Approximation Algorithm

The baseline framework in Algorithm 2 has strong theoretical guarantees, but each call to the fixed-pair procedure is performed on the full point sequences $\mathcal{P}$ and $\mathcal{Q}$. A closer look at the proof of Lemma 7 shows that, for the quadruple corresponding to an optimal solution, retaining only the optimal subsequence $\mathcal{S}^* = sLCP(\mathcal{P}, \mathcal{Q}, \epsilon)$ and its image $f^*(\mathcal{S}^*)$ is sufficient to obtain an approximate solution satisfying Theorem 5, where $f^*$ is an optimal bijection. We therefore prune the input point sequences passed to each fixed-pair call, while ensuring that at least one optimal quadruple still retains both $\mathcal{S}^*$ and $f^*(\mathcal{S}^*)$. This leads to the improved approximation algorithm presented in Algorithm 3, which preserves the guarantee of Theorem 5.

We first justify the suffix pruning in line 4.

▶ **Lemma 8.** *Let $\mathcal{S}^* = sLCP(\mathcal{P}, \mathcal{Q}, \epsilon)$, and let $p_i$ be the first point of $\mathcal{S}^*$ in the sequence order of $\mathcal{P}$. If all candidate pairs $(p_i, \tilde{p})$ with $\tilde{p} \in \mathcal{S}^*$ and all possible image pairs in $\mathcal{Q}$ are examined, then the search finds an approximate solution satisfying Theorem 5.*

**Algorithm 3** Improved Approximation Algorithm for Protein Structure Alignment.

**Input :** Two proteins $\mathcal{P}$ ($|\mathcal{P}| = n$) and $\mathcal{Q}$ ($|\mathcal{Q}| = m$), a distance threshold $\epsilon > 0$, and an approximation factor $0 < \delta \leq 3$

**Output :** A subsequence $\hat{\mathcal{S}}$ of $\mathcal{P}$, a rigid transformation $\hat{T}$, and a bijection $\hat{f}$

1 $\hat{\mathcal{S}} \leftarrow (), \hat{T} \leftarrow$ identity transformation, $\hat{f} \leftarrow$ empty bijection;
2 **for** $i_1 = 1$ **to** $n$ **do**
3  **for** $i_2 = i_1 + 1$ **to** $n$ **do**
4  $\mathcal{P}' \leftarrow (p_{i_1}, p_{i_1+1}, \ldots, p_n)$ ; // Lemma 8
5  $\mathcal{P}' \leftarrow \text{filter}(\mathcal{P}',$ condition: $D(p_{i_1}, \cdot) \leq D(p_{i_1}, p_{i_2}))$ ; // Lemma 9
6  **for** $(q_{j_1}, q_{j_2}) \in \mathcal{Q} \times \mathcal{Q}$ *with* $j_1 \neq j_2$ **do**
7  $\mathcal{Q}' \leftarrow \text{filter}(\mathcal{Q},$ condition: $D(q_{j_1}, \cdot) \leq D(p_{i_1}, p_{i_2}) + 2\epsilon)$ ; // Lemma 10
8  Calculate $\mathcal{S}_c, T_c, f_c$ from $(\mathcal{P}', \mathcal{Q}', \epsilon, \delta, (p_{i_1}, p_{i_2}), (q_{j_1}, q_{j_2}))$ ; // Lemma 7
9  **if** $|\mathcal{S}_c| > |\hat{\mathcal{S}}|$ **then**
10  $\hat{\mathcal{S}} \leftarrow \mathcal{S}_c, \hat{T} \leftarrow T_c, \hat{f} \leftarrow f_c$;
11 **return** $\hat{\mathcal{S}}$, $\hat{T}$, $\hat{f}$;

**Proof.** Since $p_i \in \mathcal{S}^*$, there exists a radial pair $(p_i, p')$ of $\mathcal{S}^*$ by definition. Because $p_i$ is the first point of $\mathcal{S}^*$, the point $p'$ is contained in the suffix $(p_i, p_{i+1}, \ldots, p_n)$. Let $q = f^*(p_i)$ and $q' = f^*(p')$. Since all image pairs in $\mathcal{Q}$ are examined, the pair $(q, q')$ is considered. Thus, the search includes a quadruple satisfying Lemma 7, and the corresponding fixed-pair call returns an approximate solution satisfying Theorem 5. ◀

Next, we justify the distance filter in line 5.

▶ **Lemma 9.** *If $(p, p')$ is a radial pair of $\mathcal{S}^*$, then no point $p'' \in \mathcal{S}^*$ has $D(p, p'') > D(p, p')$.*

**Proof.** By definition, $p'$ is a farthest point from $p$ in $\mathcal{S}^*$. ◀

Therefore, when $(p_{i_1}, p_{i_2})$ is the radial pair used by an optimal quadruple, line 5 removes only points that cannot belong to $\mathcal{S}^*$.

Finally, we justify the pruning of $\mathcal{Q}$ in line 7.

▶ **Lemma 10.** *Let $(p, p')$ be a radial pair of $\mathcal{S}^*$, and let $f^*$ be an optimal bijection. If $f^*(p) = q$, then every point $q'' \in f^*(\mathcal{S}^*)$ satisfies $D(q, q'') \leq D(p, p') + 2\epsilon$.*

**Proof.** Let $T^*$ be an optimal rigid transformation, and $p'' \in \mathcal{S}^*$ satisfy $f^*(p'') = q''$. Since $(p, p')$ is a radial pair of $\mathcal{S}^*$, we have $D(p, p'') \leq D(p, p')$. Moreover, $D(T^*(p), q) \leq \epsilon$ and $D(T^*(p''), q'') \leq \epsilon$. Because $T^*$ is rigid, $D(T^*(p), T^*(p'')) = D(p, p'')$. By the triangle inequality, $D(q, q'') \leq D(q, T^*(p)) + D(T^*(p), T^*(p'')) + D(T^*(p''), q'') \leq D(p, p') + 2\epsilon$. ◀

Combining Lemmas 8–10, consider the iteration in which $p_{i_1}$ is the first point of $\mathcal{S}^*$, $p_{i_2}$ is a radial point of $p_{i_1}$ in $\mathcal{S}^*$, and $(q_{j_1}, q_{j_2}) = (f^*(p_{i_1}), f^*(p_{i_2}))$. In this iteration, the reduced sequences still contain $\mathcal{S}^*$ and $f^*(\mathcal{S}^*)$, so the fixed-pair call satisfies Lemma 7. Therefore, Algorithm 3 preserves the guarantee of Theorem 5, while reducing the practical search space.

In the next section, we evaluate these approximation algorithms under the original threshold $\epsilon$ of the Protein Structure Alignment problem, counting only alignments with $D(T(\mathcal{S}), f(\mathcal{S})) \leq \epsilon$. Thus, when needed, an algorithm with approximation factor $\delta$ is run with threshold $\epsilon/(1 + \delta)$; this is consistent with the evaluation protocol of [13, 19]. In our implementation of the proposed pruning framework, we use AlignFastLCS rather than AlignFastLIS as the underlying algorithm from [19], since AlignFastLCS was generally shown to be faster in their experiments.

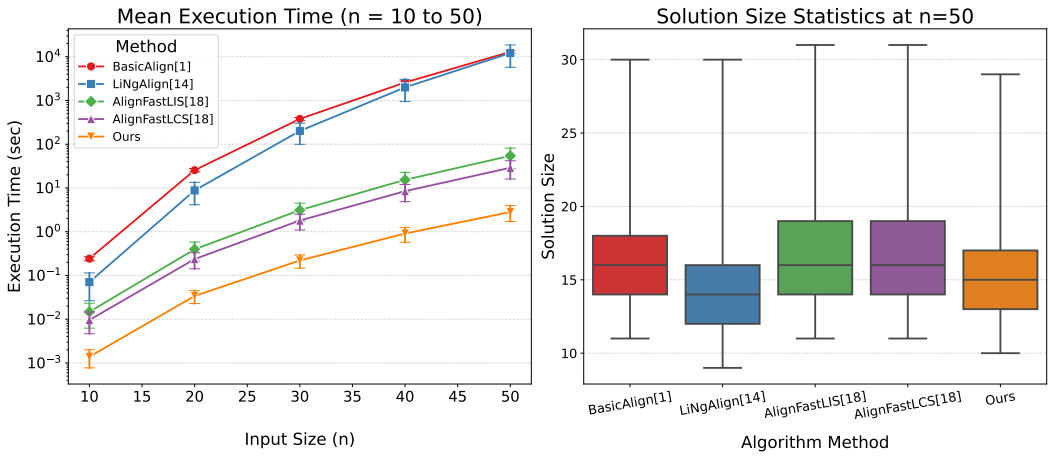



**Figure 2** Results of Experiment 1 (equal lengths). **(Left)** Mean execution time for input sizes $n \in \{10, 20, 30, 40, 50\}$. The vertical axis is logarithmic in seconds, and error bars show the standard deviation over $10^3$ trials. **(Right)** Distribution of alignment sizes at $n = 50$. Each box plot shows the median, interquartile range, and whiskers over $10^3$ trials.

**Table 1** Detailed results of Experiment 1, showing average execution time (s) and mean alignment size for $n = 10$ to 50.

| | BasicAlign[1] | | LiNgAlign[14] | | AlignFastLIS[19] | | AlignFastLCS[19] | | Ours | |
|---|---|---|---|---|---|---|---|---|---|---|
| $n$*) | Time | Size | Time | Size | Time | Size | Time | Size | Time | Size |
| 10 | 0.24261 | 6.47 | 0.07036 | 6.57 | 0.01473 | 6.61 | 0.00958 | 6.61 | 0.00139 | 6.24 |
| 20 | 25.4969 | 10.08 | 8.78990 | 9.89 | 0.39586 | 10.26 | 0.23665 | 10.26 | 0.03394 | 9.50 |
| 30 | 379.052 | 12.60 | 200.972 | 11.89 | 3.11385 | 12.81 | 1.80055 | 12.81 | 0.22019 | 11.74 |
| 40 | 2578.34 | 14.84 | 1993.46 | 13.65 | 15.3602 | 15.07 | 8.45435 | 15.07 | 0.90889 | 13.80 |
| 50 | 12921.2 | 16.46 | 12157.9 | 14.66 | 54.3501 | 16.69 | 28.9667 | 16.69 | 2.82026 | 15.26 |

*) $n$ is the length of each input protein, where $n = m$.

## 5 Experiments and Results

To evaluate the practical performance of our approach, we implemented the proposed algorithm in `C++` and compared it with baseline approximation algorithms from previous studies [1, 14, 19]. The experiments used protein structures obtained from the Protein Data Bank (PDB) [6] on May 9, 2026. Our implementation is available at `https://github.com/masat03110/ProteinStructureAlignmentPlus`.

All experiments were conducted on the SHIROKANE supercomputer at the Human Genome Center, The University of Tokyo. Each node is equipped with AMD EPYC 7713 CPUs and 512 GB of RAM, running CentOS Stream 8. The source code was compiled with GCC 13.2.0 using the `-std=c++20` and `-O3` flags. Each task was executed as a single-core job with 2.1 GB of memory using the `qsub` job scheduler.

### 5.1 Design and Execution

We conducted two experiments under the same basic settings: distance threshold $\epsilon = 2.0$ Å, approximation factor $\delta = 3$, and equal input sizes ($n = m$). For each parameter setting, we performed $10^3$ trials. In each trial, we sampled random consecutive segments of C$\alpha$ atoms from PDB proteins and measured both execution time and alignment size.

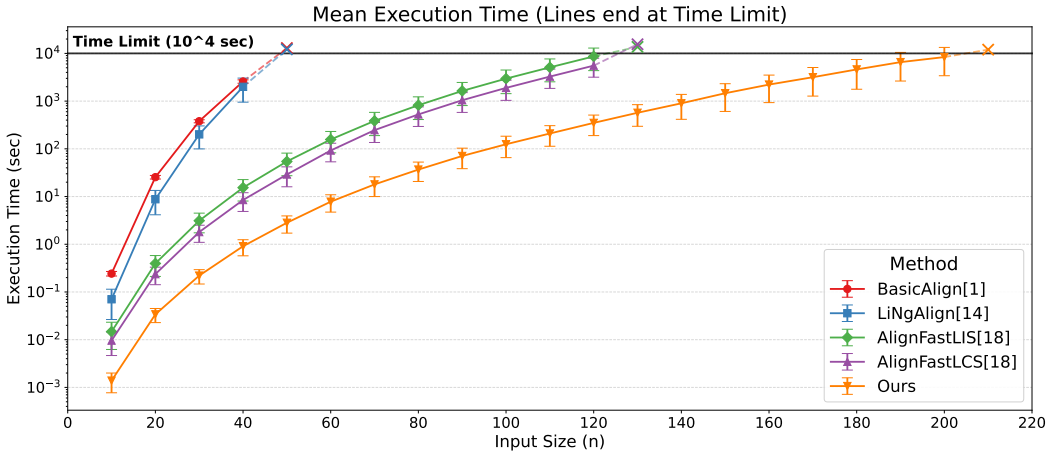



**Figure 3** Results of Experiment 2 (scalability under a time limit). The graph shows mean execution time as a function of input size $n$, with the vertical axis on a logarithmic scale in seconds. Each method was evaluated while increasing $n$ until its mean execution time exceeded $10^4$ seconds. Dashed lines with an 'x' mark indicate where evaluation of a method was stopped under this rule. Error bars show the standard deviation over $10^3$ trials.

**Table 2** Detailed results of Experiment 2 for selected input sizes, showing average execution time (s) and mean alignment size.

| | BasicAlign[1] | | LiNgAlign[14] | | AlignFastLIS[19] | | AlignFastLCS[19] | | Ours | |
|---|---|---|---|---|---|---|---|---|---|---|
| $n$ | Time | Size | Time | Size | Time | Size | Time | Size | Time | Size |
| 120 | –*) | –*) | –*) | –*) | 8647.04 | 25.61 | 5540.95 | 25.61 | 349.624 | 23.39 |
| 130 | –*) | –*) | –*) | –*) | 13687.5 | 26.64 | 15548.6 | 26.64 | 569.733 | 24.29 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| 200 | –*) | –*) | –*) | –*) | –*) | –*) | –*) | –*) | 8375.27 | 29.67 |
| 210 | –*) | –*) | –*) | –*) | –*) | –*) | –*) | –*) | 11986.1 | 30.44 |

*) "–" indicates that the method was not evaluated because its mean execution time had already exceeded $10^4$ seconds at a smaller input size.

**Experiment 1: Equal Lengths.** All methods were evaluated for input sizes $n \in \{10, 20, 30, 40, 50\}$. The average execution times and alignment sizes are shown in Figure 2 and Table 1.

**Experiment 2: Scalability under a Time Limit.** We evaluated scalability for input sizes from $n = 10$ to 220 in increments of 10, increasing $n$ until the mean execution time exceeded $10^4$ seconds. Once the mean execution time of a method exceeded $10^4$ seconds at some input size, we did not evaluate that method for larger inputs. The results are shown in Figure 3 and Table 2.

## 5.2 Observations and Analysis

In Experiment 1, our method achieved a substantial speedup over the baseline algorithms while maintaining comparable alignment quality. As shown in Table 1, at $n = 50$ our method was more than 10 times faster than AlignFastLCS, the fastest baseline, and over 4500 times faster than BasicAlign. Although its mean alignment size was slightly smaller than those

of BasicAlign, AlignFastLIS, and AlignFastLCS, the difference was modest. The box plots in Figure 2 (Right) also show that, at $n = 50$, the alignment sizes produced by our method remain close to those of the high-accuracy baselines. This is consistent with the fact that our pruning preserves the same theoretical guarantee as the framework of Tsukahara and Shibuya [19].

Experiment 2 further demonstrates the scalability advantage of our method. As shown in Figure 3 and Table 2, AlignFastLIS and AlignFastLCS become impractical around $n = 120$–$130$: at $n = 130$, they require 13687.5 and 15548.6 seconds on average, respectively. In contrast, our method finishes in 349.624 seconds at $n = 120$ and 569.733 seconds at $n = 130$, corresponding to 5.83 and 9.50 minutes. Thus, at $n = 130$, our method achieves over a 24-fold speedup over the fastest baseline. Moreover, our method remains applicable to larger inputs, processing $n = 200$ and $n = 210$ in 8375.27 and 11986.1 seconds, or 2.33 and 3.33 hours on average, respectively.

These improvements are important in light of the PDB length distribution discussed in the introduction. As shown in Figure 1, the highest protein entity-length bin is (120, 130] residues, and the 25%, 50%, and 75% percentiles correspond to $L = 130$, $L = 210$, and $L = 340$, respectively. Under our experimental setting, the baseline methods become too slow around the first quartile of this distribution, whereas our method reaches approximately the median length. These results indicate that the proposed pruning substantially expands the practically accessible range of guaranteed protein structure alignment from roughly the shortest 25% of PDB entities to instances representative of nearly 50% of them.

## 6 Conclusion and Future Work

In this paper, we studied the Protein Structure Alignment problem through the sequential LCP formulation under the bottleneck distance. On the theoretical side, we adapted the framework of Choi and Goyal and improved the resulting exact algorithm from $O(n^{14})$ time to $O(\Delta_\epsilon n^{13} \log n)$ time by exploiting the bounded-density property of protein structures. On the practical side, we proposed a filtering technique for the guaranteed approximation framework of Li and Ng and Tsukahara and Shibuya. The proposed method preserves the theoretical approximation guarantee while substantially reducing the input size of each fixed-pair computation.

Experiments on PDB data demonstrated that our method maintains solution quality comparable to existing guaranteed algorithms for small instances with $n = 10, \ldots, 50$, while achieving much better scalability. In particular, our method finished in 5.83 and 9.50 minutes on average at $n = 120$ and $n = 130$, respectively, and achieved over a 24-fold speedup at $n = 130$, the most frequent size range in the PDB. Moreover, it processed instances with $n = 200$ and $n = 210$ in 2.33 and 3.33 hours on average, respectively, thereby reaching nearly the median length of PDB entities and covering nearly 50% of them. These results suggest that the proposed pruning makes theoretically guaranteed protein structure alignment more practical for real biological datasets.

Several directions remain for future work. First, the enumeration over candidate radial pairs and image pairs is highly parallelizable, and parallel implementation may further extend the applicable input size. Second, randomized sampling or heuristic prioritization of promising candidate pairs may provide additional speedups, especially when one can trade strict worst-case guarantees for practical efficiency. Finally, integrating the proposed approach with existing structural search pipelines may make it useful for broader database-scale analyses in structural bioinformatics.

## References


1 Tatsuya Akutsu. Protein structure alignment using dynamic programing and iterative improvement. *IEICE Transactions on Information*, E79-D(12):1629–1636, 1996. URL: https://globals.ieice.org/en_transactions/information/10.1587/e79-d_12_1629/_p.

2 Christoph Ambühl, Samarjit Chakraborty, and Bernd Gärtner. Computing largest common point sets under approximate congruence. In Mike S. Paterson, editor, *Algorithms - ESA 2000*, pages 52–64. Springer Berlin Heidelberg, 2000. doi:10.1007/3-540-45253-2_6.

3 K. S. Arun, T. S. Huang, and S. D. Blostein. Least-squares fitting of two 3-d point sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-9(5):698–700, 1987. doi:10.1109/TPAMI.1987.4767965.

4 Saugata Basu, Richard Pollack, and Marie-Françoise Roy. *Algorithms in Real Algebraic Geometry*, volume 10 of *Algorithms and Computation in Mathematics*. Springer, 2 edition, 2006. doi:10.1007/3-540-33099-2.

5 Rudolf Bayer. Symmetric binary b-trees: Data structure and maintenance algorithms. *Acta Informatica*, 1(4):290–306, December 1972. doi:10.1007/BF00289509.

6 Helen M. Berman, John Westbrook, Zukang Feng, Gary Gilliland, T. N. Bhat, Helge Weissig, Ilya N. Shindyalov, and Philip E. Bourne. The protein data bank. *Nucleic Acids Research*, 28(1):235–242, January 2000. doi:10.1093/nar/28.1.235.

7 Nathalie S. Boutonnet, Marianne J. Rooman, Maria-Elena Ochagavia, Jean Richelle, and Shoshana J. Wodak. Optimal protein structure alignments by multiple linkage clustering: application to distantly related proteins. *Protein Engineering, Design and Selection*, 8(7):647–662, July 1995. doi:10.1093/protein/8.7.647.

8 Samarjit Chakraborty and Somenath Biswas. Approximation algorithms for 3-d common substructure identification in drug and protein molecules. In Frank Dehne, Jörg-Rüdiger Sack, Arvind Gupta, and Roberto Tamassia, editors, *Algorithms and Data Structures*, pages 253–264. Springer Berlin Heidelberg, 1999. doi:10.1007/3-540-48447-7_26.

9 Vicky Choi and Navin Goyal. A combinatorial shape matching algorithm for rigid protein docking. In Suleyman Cenk Sahinalp, S. Muthukrishnan, and Ugur Dogrusoz, editors, *Combinatorial Pattern Matching*, pages 285–296, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. doi:10.1007/978-3-540-27801-6_21.

10 Vicky Choi and Navin Goyal. An efficient approximation algorithm for point pattern matching under noise. In José R. Correa, Alejandro Hevia, and Marcos Kiwi, editors, *LATIN 2006: Theoretical Informatics*, pages 298–310. Springer Berlin Heidelberg, 2006. doi:10.1007/11682462_30.

11 Lars Holm, Christos Ouzounis, Chris Sander, Georg Tuparev, and Gert Vriend. A database of protein structure families with common folding motifs. *Protein Science*, 1(12):1691–1698, 1992. doi:10.1002/pro.5560011217.

12 Shuai Cheng Li, Dongbo Bu, Jinbo Xu, and Ming Li. Finding largest well-predicted subset of protein structure models. In Paolo Ferragina and Gad M. Landau, editors, *Combinatorial Pattern Matching*, pages 44–55, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. doi:10.1007/978-3-540-69068-9_7.

13 Shuai Cheng Li, Dongbo Bu, Jinbo Xu, and Ming Li. Finding nearly optimal gdt scores. *Journal of Computational Biology*, 18(5):693–704, 2011. doi:10.1089/cmb.2010.0123.

14 Shuai Cheng Li and Yen Kaow Ng. On protein structure alignment under distance constraint. *Theoretical Computer Science*, 412(32):4187–4199, 2011. Algorithms and Computation. doi:10.1016/j.tcs.2010.11.045.

15 John Moult, Krzysztof Fidelis, Andriy Kryshtafovych, Burkhard Rost, and Anna Tramontano. Critical assessment of methods of protein structure prediction - round viii. *Proteins*, 77(Suppl 9):1–4, 2009. doi:10.1002/prot.22589.

16 John Moult, Jan T. Pedersen, Richard Judson, and Krzysztof Fidelis. A large-scale experiment to assess protein structure prediction methods. *Proteins: Structure, Function, and Bioinformatics*, 23(3):ii–iv, 1995. doi:10.1002/prot.340230303.

**17** Aleksandar Poleksic. On complexity of protein structure alignment problem under distance constraint. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 9(2):511–516, 2012. doi:10.1109/TCBB.2011.133.

**18** Masahito Tsukahara and Tetsuo Shibuya. ProteinStructureAlignmentPlus. Software, swhId: swh:1:dir:8bdeadc1f4304d35619d3600f97069c17bc38192 (visited on 2026-08-10). URL: https://github.com/masat03110/ProteinStructureAlignmentPlus.

**19** Masahito Tsukahara and Tetsuo Shibuya. Efficient and accurate approximation algorithms for protein structure alignment. In *Bioinformatics Research and Applications: 21st International Symposium, ISBRA 2025, Helsinki, Finland, August 3–5, 2025, Proceedings, Part I*, pages 122–134, Berlin, Heidelberg, 2025. Springer-Verlag. doi:10.1007/978-981-95-0698-9_11.

**20** Gert Vriend and Chris Sander. Detection of common three-dimensional substructures in proteins. *Proteins*, 11(1):52–58, 1991. doi:10.1002/prot.340110107.

**21** Jinbo Xu, Feng Jiao, and Bonnie Berger. A parameterized algorithm for protein structure alignment. *Journal of Computational Biology*, 14(5):564–577, 2007. PMID: 17683261. doi:10.1089/cmb.2007.R003.

**22** Rongqing Yuan, Jing Zhang, Andriy Kryshtafovych, R. Dustin Schaeffer, Jian Zhou, Qian Cong, and Nick V. Grishin. Casp16 protein monomer structure prediction assessment. *Proteins: Structure, Function, and Bioinformatics*, 94(1):86–105, 2026. doi:10.1002/prot.70031.

**23** A. Zemla, C. Venclovas, J. Moult, and K. Fidelis. Processing and analysis of casp3 protein structure predictions. *Proteins*, 37(Suppl 3):22–29, 1999. doi:10.1002/(sici)1097-0134(1999)37:3+<22::aid-prot5>3.3.co;2-n.

**24** Adam Zemla. Lga: a method for finding 3d similarities in protein structures. *Nucleic Acids Research*, 31(13):3370–3374, July 2003. doi:10.1093/nar/gkg571.

**25** Yang Zhang and Jeffrey Skolnick. Scoring function for automated assessment of protein structure template quality. *Proteins: Structure, Function, and Bioinformatics*, 57(4):702–710, 2004. doi:10.1002/prot.20264.

# Efficient Algorithms for Pangenome Personalization

**Denys Andrukhovskyi**
Faculty of Mathematics, Physics and Informatics, Comenius University Bratislava, Slovakia

**Martin Madzin**
Faculty of Mathematics, Physics and Informatics, Comenius University Bratislava, Slovakia

**Luca Denti**
Faculty of Mathematics, Physics and Informatics, Comenius University Bratislava, Slovakia

**Tomáš Vinař**
Faculty of Mathematics, Physics and Informatics, Comenius University Bratislava, Slovakia

**Broňa Brejová**
Faculty of Mathematics, Physics and Informatics, Comenius University Bratislava, Slovakia

## Abstract

A pangenome graph is a representation of the genomes of multiple individuals of the same species. Using a pangenome graph reference instead of a single linear reference genome can increase accuracy of read mapping and downstream tasks, e.g., variant calling, but can also lead to increasing computational demands and false positives. In 2024, Sirén et al. proposed to select only parts of the pangenome mostly likely to match a studied individual, introducing the so-called personalized pangenome reference. Their algorithm is based on greedily selecting sections of paths representing individual haplotypes comprising the pangenome. In this article, we formulate the problem of pangenome personalization purely in terms of pangenome vertices and edges, as finding two paths using vertices supported by sequencing data. We provide several algorithms for solving the problem, ranging from a simple linear-time greedy algorithm with approximation ratio analysis, through dynamic programming and application of minimum-cost flow. Our implementation misses only a small percentage of vertices belonging to the studied individual and improves the sensitivity of read mapping compared to the linear reference.

**2012 ACM Subject Classification** Applied computing → Computational genomics; Theory of computation → Design and analysis of algorithms

**Keywords and phrases** pangenome graph, approximation algorithm, network flow

**Digital Object Identifier** 10.4230/LIPIcs.WABI.2026.8

**Supplementary Material** *Software (source code)*: `https://github.com/fmfi-compbio/2paths` archived at `swh:1:dir:7b3fbbcb9b5bff8245a6e1d8f001bdcf50c93523`

**Funding** *Luca Denti*: This research work has received funding from the European Union's Horizon programme under the Horizon Europe grant agreement (ASVA-CGR No. 101180581).
*Tomáš Vinař*: This research was supported in part by the Slovak Grant Agency VEGA grant 1/0567/26.
*Broňa Brejová*: This research was supported in part by the Slovak Grant Agency VEGA grant 1/0140/25.

## 1 Introduction

A *pangenome* is a representation of a collection of genomes of multiple individuals of a species [11]. Although pangenomes can be easily described as a set of strings, each one representing the haplotype of an individual of the considered population [19], they are typically represented more compactly by means of graphs [20]. In a *pangenome graph*, vertices correspond to segments of DNA sequence and edges represent adjacency relationships



© Denys Andrukhovskyi, Martin Madzin, Luca Denti, Tomáš Vinař, and Broňa Brejová;
licensed under Creative Commons License CC-BY 4.0

between these segments. A pangenome graph data structure can optionally also store the full *haplotype paths* representing the known haplotypes of the represented individuals. Pangenome graphs have been shown to decrease reference bias and improve accuracy and sensitivity of various bioinformatics tasks, such as read mapping [13, 25, 29], variant calling [10, 3, 2], and transcriptomic data analyses [26, 7].

Addressing human genome specifically, several large pangenome graphs containing many individuals were built [30, 20, 12], with the largest representing more than 1000 individuals [33]. However, larger graphs do not necessarily yield better downstream results [24, 28, 16, 6]. Indeed, the adoption of graphs containing many individuals reduces read mapping accuracy, hinders downstream analyses, and also introduces new computational challenges, such as increased memory and time requirements. To address these challenges, the idea of *personalized pangenome references* has been proposed [27, 31]. In a personalized pangenome reference, only a specific subset of the graph is selected based on the individual under investigation and downstream analyses are therefore performed using this smaller reference graph. In other words, personalizing a pangenome graph means simplifying its topology by retaining only known variants likely shared by the individual under analysis, thereby removing the additional complexity introduced by unshared and rarer variations. This approach has been shown to improve the accuracy of read mapping and variant calling, while also reducing the computational requirements [27].

Sirén et al. [27] proposed to personalize a pangenome by selecting several haplotypes (typically between 4 and 32) that best match the $k$-mers in the sequencing reads of the query individual. Each $k$-mer unique in the input pangenome is first classified as absent, heterozygous, or homozygous in the query individual based on its count in the input reads. The algorithm is a heuristic that repeatedly samples from existing haplotype paths in the pangenome, aiming for the first path to contain as many homozygous $k$-mers as possible, and subsequent paths to cover as many heterozygous $k$-mers as possible, while avoiding absent $k$-mers. The input pangenome graph is decomposed into shorter regions, and the algorithm is applied to each region, selecting haplotype paths in each region independently. If the same haplotype path is selected in an adjacent region, the two paths are merged into a single path; other haplotypes are combined into paths arbitrarily. This approach provides a practical solution for personalizing pangenome references, but it has several limitations. First, it relies on the presence of haplotype paths in the input pangenome graph, which may not always be available, especially in pangenomes constructed from fragmented assemblies. Second, the algorithm is a heuristic and does not define a proper optimization problem.

A related problem to pangenome personalization is the *haplotype path inference* problem [9, 5, 4]. The goal of this problem is to select one (in the haploid version) or two (in the diploid version) paths that best match the input reads while minimizing the number of recombinations between known paths. However, both versions of this problem are NP-hard and the algorithms proposed strictly require the haplotype paths to be present in the input graph.

In this work, we study the optimization problem of selecting two paths best matching sequencing reads from a query individual directly from a pangenome graph without considering haplotype paths. This is appropriate when haplotype paths are not available, e.g., in pangenome graphs constructed from fragmented assemblies or pangenomes from which low frequency vertices have been filtered out, fragmenting full-length haplotype paths. In our formulation, we consider directed acyclic graphs (DAGs) representing a pangenome graph consisting of a single connected component (e.g., a single chromosome). We assign a score to each vertex of the graph which depends on the number of times the vertex is used in the paths and then select two paths in the graph that maximize the overall score (Section 2).

We provide both exact and approximate algorithms for solving this problem (Section 3), and we demonstrate their performance on real pangenome graphs built from the variant call sets provided by the Human Pangenome Reference Consortium [20] and simulated read samples (Section 4). For our experiments, we use a simple scoring scheme based on the number of $k$-mers matching vertices of the pangenome graph, similar to scoring schemes used previously by others [9, 10].

## 2 Problem Definition

In this article, we consider a pangenome in the form of a directed acyclic graph $G = (V, E)$ with a single source vertex $s$ and target vertex $t$. Each vertex is labeled with a string over alphabet $\{A, C, G, T\}$. Unless specified otherwise, a *path* refers to a directed path from $s$ to $t$ in $G$. Each such path represents a potential genomic sequence obtained by concatenating the labels of vertices on the path.

Our goal will be to select two (not necessarily distinct) paths in $G$ containing the variants occurring on the two haplotypes of a single individual. A vertex $u \in V$ can be included in these paths in total 0, 1, or 2 times. We define a scoring function $w(i, u)$ for $i \in \{1, 2\}$, which can either assign a reward (positive score) or a penalty (negative score) for the path passing through the vertex. The first pass of vertex $u$ invokes score $w(1, u)$, while the second pass of the same vertex invokes (possibly different) score $w(2, u)$.

We define the total score for vertex $u$ as $T(0, u) = 0$, $T(1, u) = w(1, u)$, and $T(2, u) = w(1, u) + w(2, u)$. For two paths $\pi_1, \pi_2$, represented as sequences of vertices, the score is defined as:

$$S(\pi_1, \pi_2) = \sum_{u \in V} T(|\{i : u \in \pi_i\}|, u).$$

Thus, we can formulate our *2-paths problem* as follows: given a graph $G$, vertices $s$ and $t$, and scoring function $w$, select paths $\pi_1$, $\pi_2$ maximizing score $S(\pi_1, \pi_2)$.

Some of our algorithms work for arbitrary scoring functions, while others put some restrictions on scores. In general, vertices supported by sequencing data will have positive scores, while those without support negative scores. One option of constructing the scoring function is to assign scores $w(1, u)$ based on agreement with sequencing data and then compute $w(2, u)$ in a unified fashion from $w(1, u)$. In our experiments, vertices with non-negative value of $w(1, u)$ get the score $w(2, u) = \alpha \cdot w(1, u)$ for some discounting factor $\alpha \in [0, 1]$. For $\alpha = 1$, this reduces to finding the highest scoring path using scores $w(1, u)$ and then using it twice, which does not correspond well to our goal of finding two haplotypes. For $\alpha < 1$, the second pass through a vertex will contribute less to the total score than the first pass. As a result, the second path is more likely to use other well-supported vertices and cover more of the variants present in the studied individual. For negative scores $w(1, u)$, the intuition is less clear. We decided to penalize the second pass more than the first, and thus we assign score $w(2, u) = (2 - \alpha)w(1, u)$. Note that for $\alpha = 1$, we have $w(2, u) = w(1, u)$ for both positive and negative scores, and thus the optimum solution (or one of several possible ones) will use the same path twice.

## 3 Algorithms

In this section, we introduce three algorithms to solve the 2-paths problem. We start with a simple greedy heuristic, which selects one path at a time. We show that it is not optimal and provide tight bounds on its approximation ratio (Section 3.1). The advantage of the

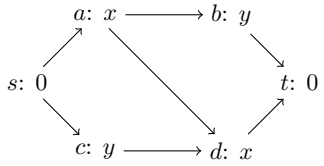



**Figure 1** An example of a graph where the greedy algorithm may produce a suboptimal solution. Each vertex is represented by its identifier and value $w(1, u)$. We set $w(2, u) = \alpha \cdot w(1, u)$ for some $\alpha \in [0, 1)$. If $x > y > \alpha x$, the greedy algorithm selects path $(s, a, d, t)$ and then either $(s, a, b, t)$ or $(s, c, d, t)$ for a total score of $2x + \alpha x + y$. The optimal solution uses paths $(s, a, b, t)$ and $(s, c, d, t)$ with a higher total score of $2x + 2y$. For example for values $x = 6, y = 5, \alpha = 1/2$, the greedy score is 20 and optimal 22.

greedy heuristic is its simplicity and efficiency, as it runs in $O(|V| + |E|)$ time. Next, we show a dynamic programming algorithm (Section 3.2) that works for an arbitrary scoring function, but its running time $O(|V| \cdot |E|)$ can be prohibitive. Fortunately, we can significantly reduce the running time by decomposing the graph into components that can be solved independently. Finally, we show a reduction to the minimum cost flow problem (Section 3.3), which can be solved in almost-linear time and also has efficient practical implementations. However, this reduction requires the scoring function to satisfy certain conditions.

## 3.1 Greedy Approximation Algorithm

The greedy algorithm we designed works as follows. We first score all vertices with $w(1, u)$ and find the longest weighted path $\pi_1$ in this graph using a simple dynamic programming in $O(|V| + |E|)$ time [8]. In the second step, we update the scores of vertices $u \in \pi_1$ to $w(2, u)$ and find the longest weighted path $\pi_2$ in this updated graph. The sum of weights of these paths is $S(\pi_1, \pi_2)$, but the two paths may not be optimal, as shown in Figure 1.

To improve this greedy heuristic, we could consider an iterative version, where after choosing $\pi_2$, we set $\pi_1 = \pi_2$, score the graph so that the vertices used in the new $\pi_1$ have score $w(2, u)$ whereas the remaining vertices have score $w(1, u)$, and then find the new $\pi_2$ as the longest weighted path with this scoring. Such iterations can be done repeatedly until there is no further improvement in the overall score.

The greedy algorithm can be applied on graphs with any scoring function, but we will now analyze its behaviour on graphs with non-negative scores that do not reward the second pass through a vertex more than the first. On such graphs, even the basic (non-iterative) greedy algorithm has a constant approximation ratio.

▷ Claim 1. Consider a constant $\alpha \in [0, 1]$ and inputs with a weight function such that $0 \leq w(2, u) \leq w(1, u)$ and $w(2, u) \geq \alpha \cdot w(1, u)$ for all $u \in V$. The greedy algorithm has an approximation ratio of at most $2/(1 + \alpha)$.

Note that the condition $w(2, u) \geq \alpha \cdot w(1, u)$ in the claim is trivially satisfied for $\alpha = 0$, and in that case the approximation ratio is 2.

Proof. Let $\pi$ be the longest weighted path in the graph scored by $w(1, u)$ and let $p$ be its score. The score of the greedy solution will be at least $p + \alpha p$, because after rescoring the graph, path $\pi$ has the score of at least $\alpha p$ and the greedy chooses $\pi_2$ with at least this score. On the other hand, the optimal solution has a score of at most $2p$ because its first path $\pi_1$ uses scores $w(1, u)$ and thus has a score of at most $p$, and the second path may use some vertices with lower scores $w(2, u)$, and therefore its score is also at most $p$.

The approximation ratio is therefore at most $2p/(p + \alpha p) = 2/(1 + \alpha)$. ◁

By a more detailed analysis, we can prove the matching upper and lower bounds on the approximation ratio of the greedy algorithm.

▶ **Theorem 2.** *Consider a constant $\alpha \in [0,1]$ and inputs with a weight function such that $0 \leq w(2,u) \leq w(1,u)$ and $w(2,u) \geq \alpha w(1,u)$ for all $u \in V$. The greedy algorithm has an approximation ratio $4/(3+\alpha)$ on such inputs.*

**Proof.** First, we will prove the **upper bound** on the approximation ratio. For any two paths $\pi$, $\pi'$, let $S(\pi') = \sum_{u \in \pi'} w(1,u)$ be the score of $\pi'$ if all its vertices are used only once, and $S_\pi(\pi') = \sum_{u \in \pi' \setminus \pi} w(1,u) + \sum_{u \in \pi \cap \pi'} w(2,u)$ be the contribution of $\pi'$ if $\pi'$ was used together with $\pi$. We will use this notation not only for full paths, but also for arbitrary sets of vertices.

For a set of vertices $U$, let $\Delta(U) = \sum_{u \in U} (w(1,u) - w(2,u))$. Since $\Delta(U)$ is the sum of non-negative terms, it is monotonic with respect to set inclusion, that is, $\Delta(U') \leq \Delta(U)$ for $U' \subseteq U$. For any paths $\pi$ and $\pi'$, we can obtain the following equality from definitions of individual scores:

$$S(\pi') = S_\pi(\pi') + \Delta(\pi \cap \pi') \tag{1}$$

From inequality $w(2,u) \geq \alpha w(1,u)$, we can get $w(1,u) - w(2,u) \leq (1-\alpha) w(1,u)$, and thus for any set $U \subseteq V$ we have

$$\Delta(U) \leq (1-\alpha) S(U). \tag{2}$$

For any three paths $\pi$, $\pi_1$ and $\pi_2$, we apply the inclusion–exclusion principle and monotonicity of $\Delta$ to obtain the following inequality:

$$\begin{aligned} \Delta(\pi \cap \pi_1) + \Delta(\pi \cap \pi_2) = \\ \Delta\left((\pi \cap \pi_1) \cup (\pi \cap \pi_2)\right) + \Delta\left((\pi \cap \pi_1) \cap (\pi \cap \pi_2)\right) = \\ \Delta((\pi_1 \cup \pi_2) \cap \pi) + \Delta(\pi_1 \cap \pi_2 \cap \pi) \leq \\ \Delta(\pi) + \Delta(\pi_1 \cap \pi_2 \cap \pi) \leq \\ \Delta(\pi) + \Delta(\pi_1 \cap \pi_2). \end{aligned} \tag{3}$$

Let $(\pi_1, \pi_2)$ be the optimal solution and $(\pi, \pi')$ be the greedy solution, where $\pi$ is the longest path with weights $w(1,u)$, and $\pi'$ is the longest path after updating the scores. As $\pi$ is the longest path, we have $S(\pi_i) \leq S(\pi)$ for $i \in \{1,2\}$. As $\pi'$ is the longest path after updating the scores, we have $S_\pi(\pi_i) \leq S_\pi(\pi')$ for $i \in \{1,2\}$. Using equation (1) for $\pi$ and $\pi_i$ we can bound $S(\pi_i)$ with $S_\pi(\pi')$ for $i \in \{1,2\}$:

$$S(\pi_i) = S_\pi(\pi_i) + \Delta(\pi \cap \pi_i) \leq S_\pi(\pi') + \Delta(\pi \cap \pi_i) \tag{4}$$

Combining formulas (1), (4), (3) and (2), we can bound the score of the optimal solution $\mathfrak{O}$ as follows

$$\begin{aligned} \mathfrak{O} = S(\pi_1) + S_{\pi_1}(\pi_2) = \\ S(\pi_1) + S(\pi_2) - \Delta(\pi_1 \cap \pi_2) \leq \\ 2S_\pi(\pi') + \Delta(\pi \cap \pi_1) + \Delta(\pi \cap \pi_2) - \Delta(\pi_1 \cap \pi_2) \leq \\ 2S_\pi(\pi') + \Delta(\pi) + \Delta(\pi_1 \cap \pi_2) - \Delta(\pi_1 \cap \pi_2) = \\ 2S_\pi(\pi') + \Delta(\pi) \leq \\ 2S_\pi(\pi') + (1-\alpha) S(\pi). \end{aligned} \tag{5}$$

Also, we will use the trivial bound from the previous claim, namely:

$$\mathfrak{O} \leq S(\pi_1) + S(\pi_2) \leq S(\pi) + S(\pi) = 2S(\pi). \tag{6}$$

The score of greedy solution is $\mathfrak{G} = S(\pi) + S_\pi(\pi')$. Now we use a linear combination of inequalities (5) and (6) to complete the proof of the upper bound.

$$2\mathfrak{O} + (1+\alpha)\mathfrak{O} \leq 2\left(2S_\pi(\pi') + (1-\alpha)S(\pi)\right) + (1+\alpha)(2S(\pi)) = 4(S_\pi(\pi') + S(\pi)) = 4\mathfrak{G}.$$

Therefore, the approximation ratio is $\frac{\mathfrak{O}}{\mathfrak{G}} \leq \frac{4}{3+\alpha}$.

To prove the **lower bound**, we can assume that $\alpha < 1$ because for $\alpha = 1$ we get an upper bound on the approximation ratio 1, which is clearly optimal. For $\alpha \in [0, 1)$, let us consider the input from Figure 1. We assume that $x > y > \alpha x \geq 0$. Under these assumptions, the cost of the optimal solution is $2x + 2y$ and the cost of greedy is $2x + \alpha x + y$. We will set $x = y(1+\varepsilon)$ for $\varepsilon > 0$. Inequality $y > \alpha x$ is trivially satisfied for $\alpha = 0$ and for $\alpha \in (0, 1)$ it holds as long as $\varepsilon < \frac{1}{\alpha} - 1$.

$$\frac{\mathfrak{O}}{\mathfrak{G}} = \frac{2x + 2y}{2x + \alpha x + y} = \frac{2y(1+\varepsilon) + 2y}{2y(1+\varepsilon) + \alpha y(1+\varepsilon) + y} = \frac{4 + 2\varepsilon}{3 + 2\varepsilon + \alpha + \alpha\varepsilon}.$$

By taking the limit $\varepsilon \to 0^+$, we get:

$$\lim_{\varepsilon \to 0^+} \frac{\mathfrak{O}}{\mathfrak{G}} = \frac{4}{3+\alpha}.$$ ◀

## 3.2 Dynamic Programming Algorithm

Now we present a dynamic programming algorithm that finds an optimal solution for any scoring function, allowing both positive and negative vertex scores without any restrictions on the relationship between $w(1, u)$ and $w(2, u)$.

Our algorithm is similar to the algorithm for finding the optimal bitonic tour credited to Jon L. Bentley [8] and the sandwich algorithm for de novo peptide sequencing [21] in the sense that it simultaneously considers and extends two paths from the starting vertex $s$.

More precisely, the subproblem used in our algorithm is indexed by a pair of vertices $u$ and $v$ where $A[u, v]$ is the score of the optimal pair of paths, both starting in $s$, the first ending in $u$ and the second ending in $v$. The trivial case is $A[s, s] = T(2, s) = w(1, s) + w(2, s)$. The score of the overall optimal solution is found in $A[t, t]$. We will fix a topological order of the vertices, which we denote by $\preceq$. Note that $A[u, v] = A[v, u]$, and thus we will treat indices of $A$ as an unordered pair. For vertices $u$ and $v$ satisfying $s \preceq u \prec v$, the following recurrence holds:

$$A[u, v] = \max_{v' \in V : (v', v) \in E} A[u, v'] + w(1, v). \tag{7}$$

The recurrence considers all possible edges entering vertex $v$: one of them must be used in the optimal solution. Note that both $u$ and $v'$ precede $v$ in the topological order, and thus the two paths in the optimal solution of $A[u, v']$ do not use $v$. By extending the path ending in $v'$ by edge $(v, v')$, we get two paths with score $A[u, v'] + w(1, v)$. It can also be easily seen that if the optimal solution for $A[u, v]$ uses edge $(v, v')$, then after removing this edge we must get an optimal solution for $A[u, v']$, because otherwise we could improve the paths by substituting the optimal solution. These observations imply the correctness of this recurrence.

The case $A[v, v]$ is solved by a similar recurrence which now uses $w(2, v)$, because vertex $v$ is used exactly once in the optimal solution of $A[v', v]$ and twice in $A[v, v]$:

$$A[v, v] = \max_{v' \in V:(v',v)\in E} A[v', v] + w(2, v). \tag{8}$$

This dynamic programming algorithm can be implemented by iterating over all vertices $v$ in the topological order, for each $v$ iterating over $u \preceq v$ in the topological order and computing $A[u, v]$ using these recurrences. Computation for one $v$ is proportional to $|V|$ times the indegree of $v$, leading to total time $O(|V| \cdot |E|)$ and memory $O(|V|^2)$.

Such running time would be clearly impractical for large pangenome graphs. However, both running time and memory can be significantly improved by considering only values $A[u, v]$ for which $u \preceq v$ and either $u = t$ or there is an edge $(u, w) \in E$ such that $v \preceq w$ (we say that $u$ has an edge reaching beyond $v$). Let $X$ be the set of pairs of vertices that satisfy this condition.

When computing $A[u, v]$ for $(u, v) \in X$ according to Eq. 7, we consider values of $A[u, v'] = A[v', u]$, where either $u \preceq v' \preceq v$ or $v' \preceq u \preceq v$. If $u \preceq v'$, clearly pair $(u, v') \in X$, since $u$ has an edge reaching beyond $v$ and $v' \preceq v$. In the other case, where $v' \preceq u$, pair $(v', u) \in X$, because edge $(v', v) \in E$ reaches beyond $u \preceq v$. Similar argument can be applied to computation according to Eq. 8. This implies that since $(t, t) \in X$, we only need to use pairs included in $X$ to compute the final value of $A[t, t]$.

An *articulation* in the pangenome graph is a vertex such that its removal will increase the number of connected components in the undirected version of $G$. Articulations may arise, for example, from conserved portions of the genome present without variation in all studied individuals. Articulations split the undirected version of the graph into a tree of biconnected components. We are only interested in one path of the tree connecting components containing $s$ and $t$. Each biconnected component along this path corresponds to a block of consecutive vertices in the topological order. For $(u, v) \in X$, both $u$ and $v$ must belong to the same biconnected component, because there are no edges connecting different components. Thus, the running time can be bound by $O(|E| \cdot M)$ where $M$ is the maximum size of a biconnected component in $G$.

## 3.3 Reduction to Minimum-Cost Flow

Another possibility is to reduce the problem to finding a minimum-cost flow in a directed graph. In theory, this problem can be solved in almost-linear time [32], and also practical solutions and an efficient solver implementation are available [14, 23]. In this reduction, we assume that in our scoring scheme we have $w(2, v) < w(1, v)$ for all vertices $v$. In other words, if a vertex is supported by sequencing reads, taking this vertex a second time gives a lower reward, whereas if a vertex is in conflict with sequencing reads, the penalty for taking this vertex a second time is larger than the first time.

Given a directed acyclic pangenome graph $G = (V, E)$ with source $s$ and target $t$, we replace each vertex $v \in V$ with a gadget of four vertices $v_1, v_2, v_3, v_4$ with edges $(v_1, v_3)$ and $(v_2, v_4)$. We call vertices $v_1$ and $v_2$ the input ports of vertex $v$ and vertices $v_3$ and $v_4$ its output ports. If $(u, v)$ is an edge of the original graph, we will create four edges, connecting both output ports of $u$ to input ports of $v$. The cost of all of these new edges will be 0, while cost of edges $(v_1, v_3)$ and $(v_2, v_4)$ denoted as $c_{v_1,v_3}$ and $c_{v_2,v_4}$ respectively will be derived from the scores assigned to vertex $v$ as $c_{v_1,v_3} = -w(1, v)$ and $c_{v_2,v_4} = -w(2, v)$. As a final step, we create a universal source $s'$ with edges with cost 0 connecting to input ports of $s$ and we also create a universal sink $t'$ connecting output ports of $t$ to $t'$. All edges in this network will have capacity 1 and our task is to find the minimum-cost network flow from $s'$ to $t'$ of size 2. The construction of the network is illustrated in Figure 2.

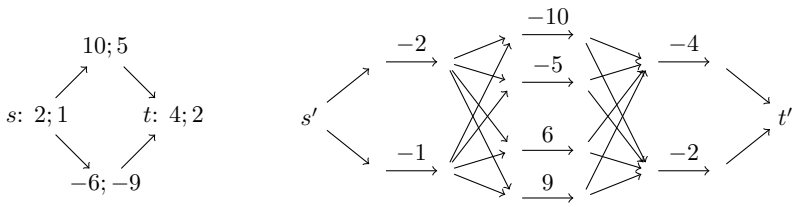



**Figure 2** An example of a graph and the resulting flow network. Each vertex $u$ in the graph on the left is labeled with $w(1, u)$ and $w(2, u)$ separated by a semicolon. The numbers in the flow network on the right show edge costs. Edges without a label have cost 0. All edges have capacity 1.

Intuitively, the resulting network flow will correspond to paths $\pi_1$ and $\pi_2$ that constitute a solution of our problem and the cost of the minimum cost flow will correspond to the score $-S(\pi_1, \pi_2)$. More precisely, if we consider two paths $\pi_1$ and $\pi_2$, we can create two edge-disjoint paths in our network, each path passing through gadgets corresponding to the vertices of the path in the original graph. If vertex $v$ is used only once, the corresponding path in the network passes through vertices $v_1$ and $v_3$ incurring contribution to the final score of $-w(1, v)$. If vertex $v$ is used twice, one of the network paths will pass through vertices $v_1$ and $v_3$ and the other one will pass through $v_2$ and $v_4$, incurring total cost of $-w(1, v) - w(2, v)$. Thus, the total cost of the corresponding network flow will sum up to $-S(\pi_1, \pi_2)$.

For the other direction, we use the integer property of minimum cost network flows: since all capacities in our network are 0 or 1 and the network flow is of size 2, there is a minimum cost flow that will use only flows of 0 or 1 on individual edges [1, Theorem 9.10]. This means that the flow can be decomposed into two edge-disjoint paths in the network, which translate to two paths $\pi_1$ and $\pi_2$ in the original graph. If vertex $v$ is used twice, the incurred cost is clearly $-w(1, v) - w(2, v)$. If the vertex is used only once, instead, the corresponding path in the network must use edge $(v_1, v_3)$, since the cost of that edge is smaller than cost of $(v_2, v_4)$ (from the assumption $w(2, v) < w(1, v)$). In this case, the contribution to the total cost is $-w(1, v)$ and the minimum cost of the network flow corresponds to $-S(\pi_1, \pi_2)$. Therefore, the cost of a minimum cost network flow corresponds to the optimal solution of our problem and the paths $\pi_1$ and $\pi_2$ can be easily reconstructed from the solution of the minimum cost network flow problem.

Note that the argument can be extended to scoring schemes for which $w(2, v) \leq w(1, v)$ (as opposed to stricter $w(2, v) < w(1, v)$). However, in such cases, one needs to consider that gadgets for vertices where the equality holds may be traversed by multiple paths with equal cost.

## 4 Experiments

In this section, we describe the experiments that we performed to evaluate the methodology described in the previous section. The goal of our experimental evaluation was twofold: (i) to assess the correctness of our algorithm and compare the two versions proposed, namely the optimal solution obtained by the minimum cost flow solver and the greedy approximation, and (ii) to evaluate the effectiveness of our methodology in subsampling a pangenome graph.

To this end, we considered the `CHM13v2.0` reference genome [22] and built real pangenome graphs from the variant call sets (v1.1) provided by the Human Pangenome Reference Consortium [20] using the `vg` toolkit [13]. In our experiments, we restricted our analysis to

single chromosomes (namely, `chr1`, `chr13`, and `chr20`) and, for each chromosome considered, we built two pangenome graphs with different numbers $n$ of individuals ($n \in \{17, 33\}$). Therefore, each graph contains $2 \times n + 1$ paths ($2 \times n$ haplotype paths and 1 reference path). We will refer to these graphs as `Full` graphs. We then selected a random individual (`HG01123`) and simulated a 20× paired-end Illumina sample from her haplotypes using `insilicoseq` [15]. Moreover, in order to simulate a real case scenario, where the sequenced individual is not actually already present in the pangenome graph, we removed her haplotype paths along with the private vertices and edges (i.e., those vertices and edges not shared with any other individual) from each `Full` graph, producing a new graph, which will be referred to as `1out` graph.

We then subsampled the graphs by solving the 2-paths problem using our implementation of the greedy and minimum-cost flow algorithms written in `C++`. The scoring of vertices for the 2-paths problem uses 31-mers computed using `KMC3` [17] from the simulated reads and is described in more detail in Section 4.1. Vertex scoring uses OpenMP for multithreading and the minimum-cost flow is solved by the OR-Tools library by Google [23]. To put our results in perspective, we compared our subsampling methodology to the subsampling implemented in the `vg` toolkit [27] (Sections 4.3 and 4.4).

## 4.1 Scoring Vertices

In our implementation, we use $k$-mer counts to assign scores $w(i, u)$ to vertices, comparing the $k$-mers represented by the pangenome graph with the $k$-mers present in the set of sequencing reads from the studied individual. Namely, let $X$ be the set of $k$-mers (we use $k = 31$) that appear in the sequencing reads of the considered individual. To handle DNA strands, we use *canonical* $k$-mers, i.e., we compare a $k$-mer with its reverse complement and use the lexicographically smaller of the two for storing and querying the set $X$. We also omit from the set $X$ the $k$-mers that appear only once in the set of reads, as these likely correspond to sequencing errors.

Let $\sigma(u)$ be the DNA sequence label of vertex $u$ and for path $\pi = (u_1, \ldots, u_n)$, let $\sigma(\pi)$ be the concatenation of strings $\sigma(u_i)$, i.e. the genomic sequence represented by this path. The general idea of our scoring is to consider a multiset of $k$-mers $X_u$ that are substrings of $\sigma(u)$ and assign a score $+1$ to each $k$-mer in $X_u$ that also appears in $X$ (which means it is supported by the reads) and score $\beta$ to each $k$-mer in $X_u$ that does not appear in $X$. The constant $\beta$ should generally be zero or negative; in the experiments we use $\beta \in \{-0.5, -1.5\}$. The overall score $w(1, u)$ is then the sum of scores of individual $k$-mers from $X_u$.

However, this simple scheme would assign zero score to all vertices with $\sigma(u)$ shorter than $k$. Also, all $k$-mers spanning vertex boundaries would be completely ignored in scoring. To address these problems, we consider all paths starting in a particular vertex $u$. Given a path $\pi' = (u, u_1, u_2, \ldots)$, we extract the first $k - 1$ characters from $\sigma(u_1, u_2, \ldots)$, i.e., the first $k - 1$ nucleotides following $\sigma(u)$ in a potential genome. We append these $k - 1$ nucleotides to $\sigma(u)$ and score the $k$-mers in the resulting string. For a given $u$ we might have multiple possible extensions of length $k - 1$. We score each of them and choose the highest score as $w(1, u)$.

In our scoring system, if the graph is a single path, every $k$-mer of the underlying string will be included in the score of exactly one vertex, namely, the vertex containing its starting position. In a more typical pangenome graph, many vertices have outdegree greater than 1. The score of a vertex then reflects the locally best path in its vicinity, but it is not guaranteed that this path is then indeed used in the final solution.

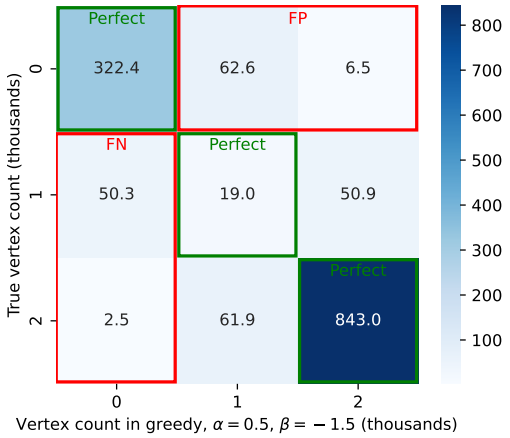



**Figure 3** The confusion matrix comparing how many times was a vertex used in the true haplotypes and the paths selected by the greedy algorithm with $\alpha = 0.5$, $\beta = -1.5$. For each combination, we show the number of vertices in thousands. The matrix elements considered as "perfect", false positives (FP) and false negatives (FN) are shown by colored frames.

We compute the score for passing a vertex $u$ a second time as follows: $w(2, u) = \alpha \cdot w(1, u)$ for $w(1, u) \geq 0$ and $w(2, u) = (2 - \alpha)w(1, u)$ for $w(1, u) < 0$. In the next section, we explore the influence of the discounting parameter $\alpha$ on the resulting pair of paths.

## 4.2 Recovery of True Vertices

In our first experiment, we ran our algorithms on the `Full` graph, which includes the vertices and edges unique to the studied individual. In this graph, we know the full haplotype paths corresponding to this individual. We will call these paths the true haplotypes. This experiment does not reflect real conditions, as when studying sequencing data from a new individual, we would not know the true haplotypes and parts of them will be missing from the graph. Nonetheless, this simple setting allows us to determine if we can recover vertices from the true haplotypes and to compare different values of parameters $\alpha$ and $\beta$ as well as the greedy and optimal algorithms. The experiments were conducted on chromosome 20, using $n = 33$ individuals.

To evaluate the results, we consider for each vertex of the graph, how many times it occurs in the true haplotypes and how many times it occurs in the two paths predicted by our algorithm. This results in a confusion matrix of vertex counts for different combinations of vertex occurrences along the paths. An example of such a confusion matrix for the greedy algorithm, $\alpha = 0.5$ and $\beta = -1.5$ is shown in Figure 3. While incorrect typing of true homozygous loci is relatively rare, many vertices that should be part of heterozygous loci (true count 1) are apparently mistakenly classified as homozygous (predicted count 0 or 2). These errors likely happen in complex regions of the graph with many nearby variants that lead to many possible $k$-mer extensions, some of which may be by chance supported by read $k$-mers originating from a different part of the chromosome. This could be addressed by employing a more complex scoring scheme.

To compare different algorithms and parameter values, we summarize the correctness of our pair of paths by three values. The first one is the fraction of vertices that are located on the main diagonal of the confusion matrix, meaning that they have the same counts in the real haplotypes and in the predicted paths (see Figure 3). We will call these vertices as *perfectly predicted* vertices. A vertex is a *false positive* (FP) if it is not on either of the

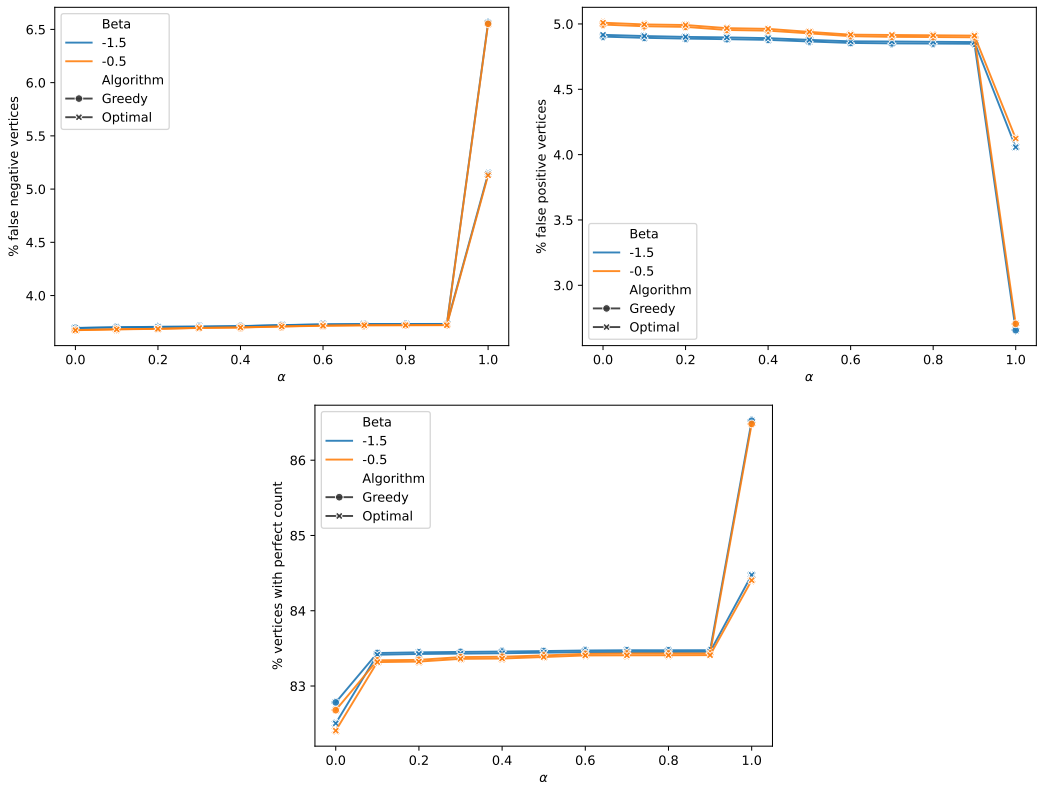



**Figure 4** The ability of our algorithms to select correct vertices from the full graph. Top: The percentage of FN (left) and FP (right) vertices out of all vertices for two algorithms and different values of $\alpha$ and $\beta$ (lower is better). Bottom: The percentage of perfectly predicted vertices out of all vertices for two algorithms and different values of $\alpha$ and $\beta$ (higher is better).

real haplotypes but is at least on one of the predicted paths. Conversely, a vertex is a *false negative* (FN) if it is in at least on one of the real haplotypes, but is not on either of the predicted paths. For pangenome personalization, a false negative is generally worse than a false positive, because exclusion of a correct vertex prevents correct alignment of the overlapping reads, whereas inclusion of spurious vertices typically leads only to slight increase in time and memory of subsequent analyses, although in some cases such vertices may also introduce ambiguity in read mapping results.

In Figure 4, we can see that the false negatives are mostly below 4% and false positives mostly below 5% of all vertices. The rates of false positives and false negatives are relatively stable for values of $\alpha$ between 0 and 0.9, but change for $\alpha = 1$. At this setting, greedy selects the same path twice, which is optimal, but the flow-based approach selects two distinct paths with the same score, thus achieving a lower number of FNs but higher FPs. The percentage of vertices with perfect counts is again stable for values of $\alpha$ from 0.3 to 0.9 and reaches the highest value of 86.5% for the greedy algorithm with $\alpha = 1$ and $\beta = -1.5$. For $\alpha < 1$, all measures shown are only slightly influenced by the choice of the algorithm. Likewise, the change in $\beta$ does not have a large influence.

Finally, Table 1 shows the approximation ratio of the greedy algorithm compared to the score obtained by the optimum flow-based algorithm, which is very close to 1 for all settings of $\alpha$. We also computed the score of the true haplotypes in our scoring scheme, obtaining a considerably worse approximation ratio than for the greedy algorithm. This suggests that

**Table 1** Approximation ratio of the greedy algorithm and the true haplotypes on the `Full` graph of chromosome 20 for 33 individuals. We show the ratio of the optimal score computed by minimum-cost flow to the score obtained by the greedy algorithm or the score obtained by scoring the paths belonging to the true haplotypes of the studied individual.

| $\alpha$ | $\beta = -1.5$ | | $\beta = -0.5$ | |
|---|---|---|---|---|
| | Greedy | True hapl. | Greedy | True hapl. |
| 0.0 | 1+4e-06 | 1+0.02 | 1+3e-06 | 1+0.02 |
| 0.3 | 1+2e-06 | 1+0.01 | 1+2e-06 | 1+0.01 |
| 0.5 | 1+1e-06 | 1+0.01 | 1+9e-07 | 1+0.01 |
| 0.7 | 1+6e-07 | 1+0.01 | 1+5e-07 | 1+0.01 |
| 0.9 | 1+2e-07 | 1+0.01 | 1+1e-07 | 1+0.01 |
| 1.0 | 1 | 1+0.01 | 1 | 1+0.01 |

the errors caused by the inexact nature of the greedy approach are negligible compared to the problems caused by the relatively simple nature of our scoring scheme which is unable to distinguish all vertices originating from the true haplotypes from vertices irrelevant for the studied individual.

## 4.3 Read Mapping Evaluation

Finally, we applied our personalization approach to pangenome graphs constructed for three different chromosomes and two different numbers of individuals. For each input, we used the greedy algorithm with $\alpha = 0.5$ and $\beta = -1.5$. After computing the two paths, we use the `vg` toolkit [13] to index the pangenome consisting only of these paths and to map the simulated sequencing reads by `vg giraffe` [28]. We measure the number of reads that perfectly map to the graph across their full length and without any errors (see Figure 5). For comparison, we also map the same reads to four other references. The first reference is the `Full` graph, which includes the studied individual. Under ideal circumstances, all errorless reads should map perfectly to this graph. Although some perfect mappings may be missed due to various mapping heuristics, this input nonetheless represents a good upper bound of what we wish to achieve in the ideal case. The second reference is the `1out` graph, which excludes the studied individual, but the 16 or 32 individuals still capture many common variants present in the human population. This corresponds to a task of read mapping to a standard non-personalized pangenome. As we can see in Figure 5, the number of perfectly mapped reads compared to `Full` decreases only by 0.3-0.4%.

Next, we consider the personalized pangenome created by diploid sampling from `vg` [27]. Here the decrease compared to `Full` is in the range of 0.4-0.7%. Our method leads to somewhat fewer perfectly mapping reads, with the decrease compared to `Full` in the range of 1.5-2.2%. This lower mapping rate is due to the false negatives observed in Section 4.2. Another contributing factor is the fact that our method is unable to correctly phase the vertices into haplotypes, whereas `vg` diploid sampling creates the pangenome from long sections of real haplotype paths stored along the pangenome graph. Since `vg giraffe` relies on these haplotypes during read mapping, lack of phasing in our paths may cause some reads to be mapped incorrectly even if perfect path is present in the graph. Finally, we mapped the reads to the linear sequence of the corresponding chromosome from the human reference genome using `BWA-MEM` [18]. Here we observe the largest drop in the perfect mapping rate, in the range of 6.6-7.9%. This comparison reveals that our personalized genome still captures much of the variation present in the studied genome.

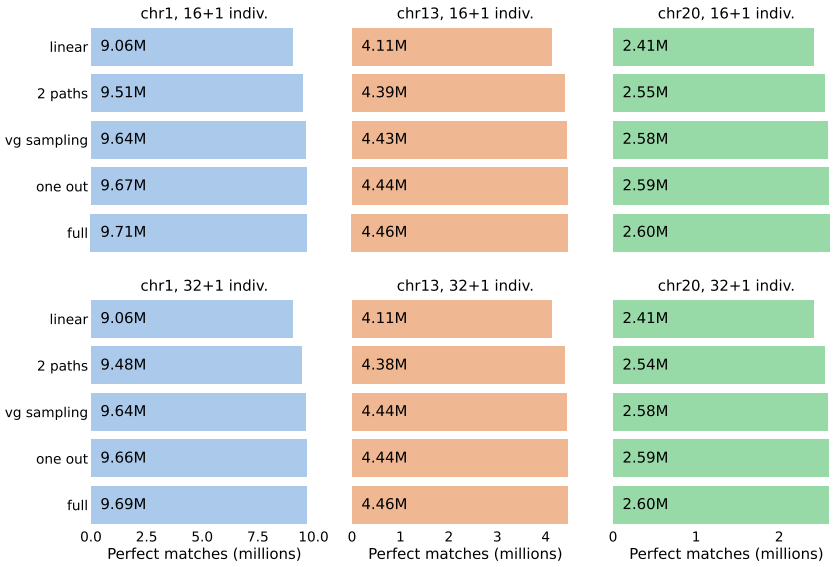



**Figure 5** The number of perfectly matching reads for different references (in million reads). Plots for each chromosome use a different scale for easier comparison of methods within a chromosome.

## 4.4 Time and Memory Benchmarks

Finally, we compared the running times and memory requirements for the inputs and approaches considered in the previous experiment. We omit the `Full` pangenome as it is similar in size to the more realistic `1out` pangenome. All tools were allowed to use 16 threads. Figure 6 shows the overall running time split into phases, including $k$-mer counting, selecting 2 paths, indexing the resulting personalized pangenome and mapping the reads to it. The overall runtime of our approach is on most inputs slightly higher than `BWA-MEM` (which works on the linear reference) and slightly lower than mapping to the `1out` pangenome. The only exception is chromosome 13 for 16 individuals, where we observed increased running time of vertex scoring, perhaps due to some thread synchronization issues. However, it is worth noting that for our algorithm, graph indexing must be performed for every new individual. In contrast, the indexing step for `BWA-MEM` and `1out` can be reused for mapping reads of multiple individuals, since the reference remains the same. The mapping phase itself is only marginally faster for the personalized graph than for the `1out` pangenome. Specifically, for 32 individuals, the mapping is only 5-10% faster. However, perhaps bigger savings could be achieved on bigger or more diverse pangenomes. On the positive side, our personalization performs significantly faster than diploid sampling by `vg`.

In Figure 7, we see the memory consumption of each phase of all tested methods. For three pangenomes, the memory bottleneck is definitely the indexing phase. As expected, personalized pangenomes use smaller memory for indexing. The memory usage of mapping on the entire graph is comparable with the memory usage of the sampling phase of `vg` personalization. Our method uses less memory and scales better with graph size. For 32 individuals, personalized genomes use about 10-14% less memory compared to `1out`.

Overall, we conclude that time and memory requirements of our implementation are similar to other pangenome-processing tools. However, at this prototype stage, our tool is not able to produce significant time and memory savings to the overall process of read mapping, particularly if reads from multiple individuals need to be mapped. Perhaps more

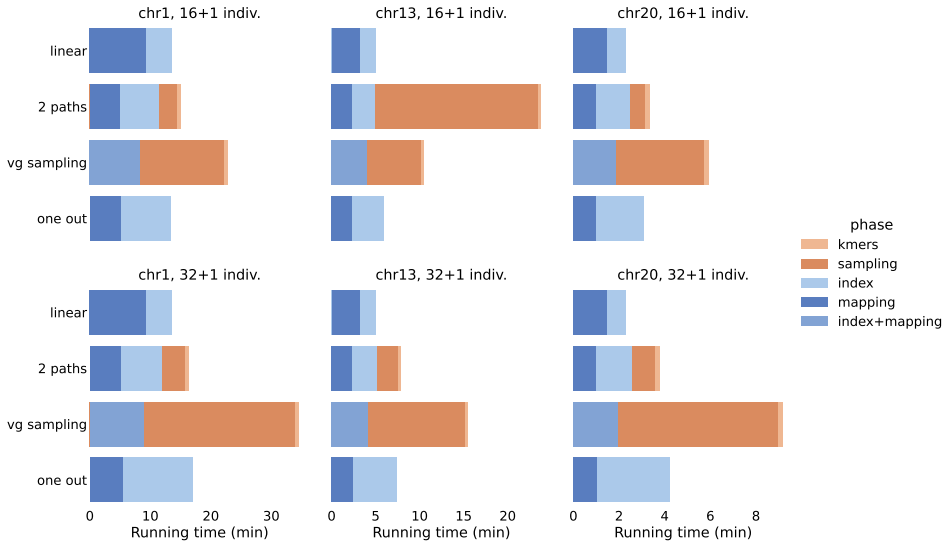



**Figure 6** The running time for different references. Plots for each chromosome use a different scale for easier comparison of methods within a chromosome. For vg sampling, indexing and mapping were done in one step.

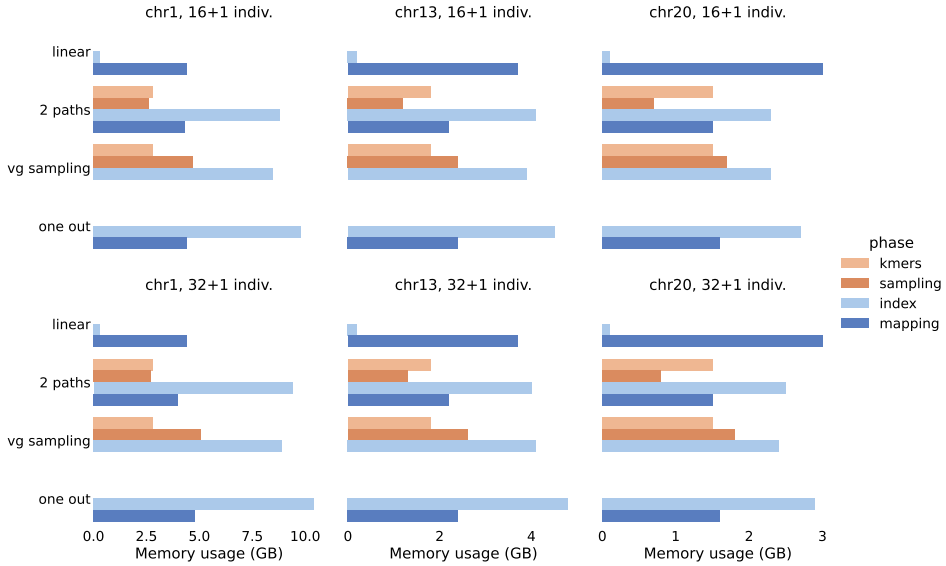



**Figure 7** The memory for different references. Plots for each chromosome use a different scale for easier comparison of methods within a chromosome. For `vg` sampling, indexing and mapping were done in one step and are shown here as indexing.

significant savings could be achieved by tighter integration of personalization and mapping, such as producing index of the personalized pangenome by selecting portions of the index for the full pangenome.

## 5 Conclusion

In this paper, we introduced a new formulation of graph personalization as finding two paths in a directed acyclic graph with optimal score, where the score for passing a vertex a second time is discounted in some way, for example by a multiplicative factor $\alpha$. We also presented three algorithms for solving this problem: a simple and efficient greedy algorithm with a provable approximation ratio, an exact dynamic programming algorithm, and a solution based on minimum-cost network flows. We implemented the first and third of these approaches and demonstrated that they can be successfully ran even on largest human chromosomes in reasonable amount of time and memory. The personalized pangenomes obtained by this method match the studied individual less well than `vg` personalization based on selecting long segments of known haplotypes. Nonetheless, they still produce a reference that matches the studied individual better than a single linear reference genome.

Advantages of our approach compared to the method implemented in `vg` are its cleaner combinatorial problem definition, which is easier to study theoretically, and applicability to graphs that do not have full haplotype paths available, such as those built from variants discovered by short-read sequencing. Our approach is also efficient: the greedy algorithm runs in linear time.

The fact that our method does not use haplotype paths is also its disadvantage in cases where these paths are available, leading to lower accuracy of our approach compared to `vg` haplotype sampling. Note that two paths recovered by our algorithm can by no means be considered as haplotypes, since we only consider the local structure of the underlying pangenome graph and make no attempt to preserve phasing.

Future work includes extending the problem beyond DAGs to at least some special classes of bidirected graphs often used to represent pangenomes. Also, extensions to finding more than two paths would be useful in situations with polyploid organisms or when "personalizing" the pangenome for a whole subpopulation.

Our approximation factor guarantees (Theorem 2) only cover the cases where all score values are non-negative. However, the scoring function actually used in the experiments assigns negative scores to unsupported vertices. Generally, it is unclear how to define approximation guarantees for scoring schemes with negative components. Perhaps, guarantees similar to Theorem 2 could be proved for cases where the final scores have high enough (positive) values. This hypothesis is empirically supported by our experiments which show that the results of the greedy algorithm are very close to optimal.

On a more practical side, better implementation and integration within `vg` or another pangenome toolkit would allow better practical applicability of our method. The method should also be tested on larger pangenomes and with real read sets. In a postprocessing step, the resulting paths could be locally phased in the areas with high density of heterozygous variants where multiple variants are spanned by input reads.

Finally, it might be beneficial to explore more complex scoring schemes. For example, tools for genome personalization or genome inference often classify $k$-mers as homozygous and heterozygous based on their counts in sequencing reads. This information could be incorporated to our scoring scheme so that the score for a homozygous $k$-mer would not be discounted during the second pass, while heterozygous $k$-mers would be discounted heavily.

## References


1 R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows: theory, algorithms, and applications*. Prentice Hall, 1993.

2 M. Asri, P. C. Chang, J. C. Mier, J. Siren, P. Eskandar, A. Kolesnikov, D. E. Cook, L. Brambrink, G. Hickey, A. M. Novak, L. Dorfman, D. R. Webster, A. Carroll, B. Paten, and K. Shafin. Pangenome-aware DeepVariant. *bioRxiv*, 2025. doi:10.1101/2025.06.05.657102.

3 S. Behera, S. Catreux, M. Rossi, S. Truong, Z. Huang, M. Ruehle, A. Visvanath, G. Parnaby, C. Roddey, V. Onuchic, A. Finocchio, D. L. Cameron, A. English, S. Mehtalia, J. Han, R. Mehio, and F. J. Sedlazeck. Comprehensive genome analysis and variant detection at scale using DRAGEN. *Nat Biotechnol*, 43(7):1177–1191, 2025.

4 G. Chandra, W. T. Doan, and D. Gibney. Haplotype-resolved diploid genome inference on pangenome graphs. *bioRxiv*, 2025. doi:10.1101/2025.11.26.690754.

5 G. Chandra, H. Helal, S. Scholz, A. T. Dilthey, D. Gibney, and C. Jain. Integer programming framework for pangenome-based genome inference. In *International Conference on Research in Computational Molecular Biology (RECOMB)*, pages 405–408. Springer, 2025.

6 R. Chikhi, Y. Dufresne, and P. Medvedev. Constructing and personalizing population pangenome graphs. *Nat Methods*, 21(11):1980–1981, 2024.

7 S. Ciccolella, D. Cozzi, G. Della Vedova, S. N. Kuria, P. Bonizzoni, and L. Denti. Differential quantification of alternative splicing events on spliced pangenome graphs. *PLOS Computational Biology*, 20(12):e1012665, 2024.

8 T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

9 A. Dilthey, C. Cox, Z. Iqbal, M. R. Nelson, and G. McVean. Improved genome inference in the MHC using a population reference graph. *Nature Genetics*, 47(6):682–688, 2015.

10 J. Ebler, P. Ebert, W. E. Clarke, T. Rausch, P. A. Audano, T. Houwaart, Y. Mao, J. O. Korbel, E. E. Eichler, M. C. Zody, A. T. Dilthey, and T. Marschall. Pangenome-based genome inference allows efficient and accurate genotyping across a wide spectrum of variant classes. *Nature Genetics*, 54(4):518–525, 2022.

11 J. M. Eizenga, A. M. Novak, J. A. Sibbesen, S. Heumos, A. Ghaffaari, G. Hickey, X. Chang, J. D. Seaman, R. Rounthwaite, J. Ebler, M. Rautiainen, S. Garg, B. Paten, T. Marschall, J. Siren, and E. Garrison. Pangenome Graphs. *Annu Rev Genomics Hum Genet*, 21:139–162, 2020.

12 Y. Gao, X. Yang, H. Chen, X. Tan, Z. Yang, L. Deng, B. Wang, S. Kong, S. Li, Y. Cui, C. Lei, Y. Wang, Y. Pan, S. Ma, H. Sun, X. Zhao, Y. Shi, Z. Yang, D. Wu, S. Wu, X. Zhao, B. Shi, L. Jin, Z. Hu, Y. Lu, J. Chu, K. Ye, and S. Xu. A pangenome reference of 36 Chinese populations. *Nature*, 619(7968):112–121, 2023.

13 Erik Garrison, Jouni Sirén, Adam M. Novak, Glenn Hickey, Jordan M. Eizenga, Eric T. Dawson, William Jones, Shilpa Garg, Charles Markello, Michael F. Lin, Benedict Paten, and Richard Durbin. Variation graph toolkit improves read mapping by representing genetic variation in the reference. *Nature Biotechnology*, 36(9):875–879, 2018.

14 A. V. Goldberg and R. E. Tarjan. Finding minimum-cost circulations by successive approximation. *Mathematics of Operations Research*, 15(3):430–466, 1990. doi:10.1287/MOOR.15.3.430.

15 H. Gourlé, O. Karlsson-Lindsjö, J. Hayer, and E. Bongcam-Rudloff. Simulating Illumina metagenomic data with InSilicoSeq. *Bioinformatics*, 35(3):521–522, 2019. doi:10.1093/BIOINFORMATICS/BTY630.

16 C. Jain, N. Tavakoli, and S. Aluru. A variant selection framework for genome graphs. *Bioinformatics*, 37(S1):i460–i467, 2021.

17 M. Kokot, M. Długosz, and S. Deorowicz. KMC 3: counting and manipulating $k$-mer statistics. *Bioinformatics*, 33(17):2759–2761, 2017. doi:10.1093/BIOINFORMATICS/BTX304.

18 H. Li. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. *arXiv*, 2013. doi:10.48550/arXiv.1303.3997.

19 H. Li. BWT construction and search at the terabase scale. *Bioinformatics*, 40(12):btae717, 2024. doi:10.1093/BIOINFORMATICS/BTAE717.

20 W. W. Liao, M. Asri, J. Ebler, et al. A draft human pangenome reference. *Nature*, 617(7960):312–324, 2023.

21 B. Ma, K. Zhang, and C. Liang. An effective algorithm for peptide de novo sequencing from MS/MS spectra. *Journal of Computer and System Sciences*, 70(3):418–430, 2005. doi:10.1016/J.JCSS.2004.12.001.

22 S. Nurk, S. Koren, A. Rhie, M. Rautiainen, A. V. Bzikadze, A. Mikheenko, M. R. Vollger, N. Altemose, L. Uralsky, A. Gershman, et al. The complete sequence of a human genome. *Science*, 376(6588):44–53, 2022.

23 L. Perron and V. Furnon. OR-Tools v9.15, 2026. URL: https://developers.google.com/optimization/.

24 J. Pritt, N. C. Chen, and B. Langmead. FORGe: prioritizing variants for graph genomes. *Genome Biol*, 19(1):220, 2018.

25 M. Rautiainen and T. Marschall. GraphAligner: rapid and versatile sequence-to-graph alignment. *Genome Biol*, 21(1):253, 2020.

26 J. A. Sibbesen, J. M. Eizenga, A. M. Novak, J. Sirén, X. Chang, E. Garrison, and B. Paten. Haplotype-aware pantranscriptome analyses using spliced pangenome graphs. *Nature Methods*, 20(2):239–247, 2023.

27 J. Sirén, P. Eskandar, M. T. Ungaro, et al. Personalized pangenome references. *Nature Methods*, 21(11):2017–2023, 2024.

28 J. Siren, E. Garrison, A. M. Novak, B. Paten, and R. Durbin. Haplotype-aware graph indexes. *Bioinformatics*, 36(2):400–407, 2020. doi:10.1093/BIOINFORMATICS/BTZ575.

29 J. Siren, J. Monlong, X. Chang, A. M. Novak, J. M. Eizenga, C. Markello, J. A. Sibbesen, G. Hickey, P. C. Chang, A. Carroll, N. Gupta, S. Gabriel, T. W. Blackwell, A. Ratan, K. D. Taylor, S. S. Rich, J. I. Rotter, D. Haussler, E. Garrison, and B. Paten. Pangenomics enables genotyping of known structural variants in 5202 diverse genomes. *Science*, 374(6574):abg8871, 2021.

30 H. S. Tetikol, D. Turgut, K. Narci, G. Budak, O. Kalay, E. Arslan, S. Demirkaya-Budak, A. Dolgoborodov, D. Kabakci-Zorlu, V. Semenyuk, A. Jain, and B. N. Davis-Dusenbery. Pan-African genome demonstrates how population-specific genome graphs improve high-throughput sequencing data analysis. *Nat Commun*, 13(1):4384, 2022.

31 K. Vaddadi, M. J. Lin, S. Majidian, T. Mun, and B. Langmead. Minimizing reference bias with an imputed personalized reference. *Genome Res*, 36(4):740–753, 2026.

32 J. van den Brand, L. Chen, R. Kyng, Y. P. Liu, R. Peng, M. P. Gutenberg, S. Sachdeva, and A. Sidford. A deterministic almost-linear time algorithm for minimum-cost flow. In *64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 503–514. IEEE, 2023.

33 Y. Wang, Z. Duan, D. Chen, D. Shi, Y. Ding, Z. Wang, B. Li, Z. Wang, M. Guo, W. Yang, J. Hou, W. Chen, Y. Guo, W. Wei, Y. Cao, X. Sun, W. Bai, M. Lu, T. Qi, X. Shen, and J. Yang. The 1000 Chinese Pangenome empowers medical and population genetics. *Nature*, 654:121–130, 2026.

# Quantum Closest-Pair Search for Biological Sequences via k-Mer Distribution Statistics

**Zhezheng Xander Song**
Ray and Stephanie Lane Computational Biology Department,
Carnegie Mellon University, Pittsburgh, PA, USA

**Carl Kingsford**[1]
Ray and Stephanie Lane Computational Biology Department,
Carnegie Mellon University, Pittsburgh, PA, USA

## Abstract

Finding highly similar pairs of biological sequences is a fundamental task in bioinformatics. For alignment-free $k$-mer distributional similarities, the induced feature space is high-dimensional and lacks the low-dimensional geometric structure used by classical exact closest-pair algorithms. Thus, for a collection of $N$ sequences in the pairwise-score setting, exhaustive evaluation over the $\binom{N}{2}$ candidate pairs is the natural classical baseline.

We present the first quantum framework targeting alignment-free closest-pair search in biological sequence collections using distributional $k$-mer statistics. The central technical contribution is the construction of a coherent pairwise-score estimation circuit for this similarity measure. It encodes empirical $k$-mer distributions as square-root amplitude states and provides a sparse prefix-tree construction for preparing these states, under which the state overlap is exactly the Bhattacharyya coefficient. Standard SWAP-test and quantum-amplitude-estimation subroutines provide a coherent bounded-precision estimator for the squared Bhattacharyya overlap.

We analyze maximum finding under an explicit assumption that a fixed $\varepsilon$-resolved total order over all legal pairs admits an efficient clean coherent implementation. Under this assumption, the procedure returns, with probability at least 2/3, a pair whose squared Bhattacharyya score is within $\varepsilon$ of the optimal score, using $O(N)$ expected comparison-oracle calls. If the optimal score is separated from every strictly suboptimal score by more than $\varepsilon$, the returned pair is exactly optimal.

Combining this comparison-order assumption with an idealized qRAM-style data-access model gives the conditional sequential gate complexity $\widetilde{O}\left(\frac{NL}{\varepsilon}\right)$, whereas explicit multiplexed indexed loading gives $\widetilde{O}\left(\frac{N^2L}{\varepsilon}\right)$. We also provide a proof-of-concept Q# implementation that integrates coherent indexed loading, SWAP-test-based score estimation, finite-precision marking, and Grover-style search, providing circuit-level validation of the main computational components.

**2012 ACM Subject Classification** Theory of computation → Quantum query complexity; Applied computing → Molecular sequence analysis; Theory of computation → Quantum computation theory; Theory of computation → Design and analysis of algorithms

**Keywords and phrases** quantum algorithms, closest-pair search, k-mer statistics, Bhattacharyya coefficient, Dürr–Høyer maximum finding, quantum amplitude estimation

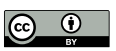
**Digital Object Identifier** 10.4230/LIPIcs.WABI.2026.9

**Supplementary Material** *Software*: `https://github.com/Kingsford-Group/tieria`

**Funding** *Carl Kingsford*: This work was supported in part by the US National Science Foundation [III-2232121], the US National Institutes of Health [R01HG012470].

**Acknowledgements** We thank Shane Elder for his careful reading of a later version, for identifying several important issues, and for suggesting substantial improvements. We also thank Frank Britto Bisso, Emmanuel Mhrous, and Yu-Hsiang Chen for their helpful comments on an earlier version of the manuscript.

[1] corresponding author



© Zhezheng Xander Song and Carl Kingsford;
licensed under Creative Commons License CC-BY 4.0





## 1 Introduction

Given a collection of biological sequences, we consider the problem of identifying the most similar pair under a prescribed sequence-similarity score. This task appears naturally in redundancy reduction [17], clustering [10], database cleaning [23], comparative genomics [20], and metagenomic analysis [19], where highly similar sequences may indicate homologous records, repeated observations, closely related strains, or sequences that should be grouped together for downstream analysis.

Traditionally, highly similar sequence pairs have been found either by direct all-pairs comparison or by scalable approximate strategies such as clustering heuristics, seed-and-extend search, sparse chaining, and sketching or locality-sensitive hashing. Direct all-pairs comparison provides a natural exact baseline but becomes expensive for large collections. Practical bioinformatics tools therefore use a variety of scalable alternatives: CD-HIT uses greedy representative-based clustering for redundancy reduction [17], USEARCH/UCLUST accelerates search and clustering through fast heuristic matching [10], VSEARCH supports large-scale clustering, dereplication, and all-vs-all comparison workflows [23], Mash uses MinHash sketches to estimate genomic distances efficiently [20], and skani estimates ANI using sparse approximate alignments and sparse chaining [26]. Minimizer-based sketching methods provide another major class of scalable sequence-comparison primitives, selecting representative $k$-mers so that sequence similarity or alignability can be estimated from compact sketches [30, 18]. These methods have been highly successful in practice, but they typically trade exact closest-pair search for filtering, approximation, or threshold-based clustering.

We analyze exact closest-pair search in a black-box pairwise-score model. The input induces $P = \binom{N}{2}$ legal candidate pairs, and an algorithm accesses a candidate pair only by evaluating its score $F(S_i, S_j)$. Without additional structure relating the scores of different pairs, exhaustive evaluation is the natural classical exact baseline. In the worst case, if a pair is left unevaluated, that pair may have a score larger than every evaluated pair. Thus, the classical exact query complexity in this model is $\Theta(P) = \Theta(N^2)$.

This black-box model is motivated by the high-dimensional nature of alignment-free $k$-mer representations. The square-root $k$-mer representations considered here lie in a space whose dimension is the number $M$ of observed $k$-mers, and we do not assume the fixed-dimensional geometric structure exploited by classical closest-pair algorithms. For example, closest-pair search in the Euclidean plane can be solved in $O(N \log N)$ time using geometric structure such as divide-and-conquer arguments or Voronoi diagrams [25]. In contrast, Karthik and Manurangsi showed that, assuming the Strong Exponential Time Hypothesis, exact monochromatic closest pair under the $\ell_p$ metric in certain high-dimensional regimes admits no strongly subquadratic classical algorithm [24]. These metric-specific hardness results provide context for why low-dimensional geometric speedups should not be assumed in our setting.

In this black-box setting, quantum maximum-finding algorithms can reduce the number of pair-score queries. The Dürr–Høyer framework reduces maximum finding to repeated quantum searches for candidates that improve on the current threshold [9]. The resulting searches use Grover-style amplitude amplification, with the BBHT procedure handling the case in which the number of improving pairs is unknown [14, 6].

This unstructured-search viewpoint is also consistent with known quantum complexity results for closest-pair problems. Aaronson et al. gave a $\widetilde{O}(N^{2/3})$ quantum algorithm for constant-dimensional Euclidean closest pair, while showing that in certain higher-dimensional

regimes a substantially better-than-Grover speedup would contradict the Quantum Strong Exponential Time Hypothesis [1]. As above, this result concerns metric closest-pair problems and is included as complexity-theoretic context rather than as a lower bound for the squared Bhattacharyya score studied here.

Many quantum algorithms that operate on large classical inputs assume a coherent data-access primitive, often modeled as quantum random access memory (qRAM). At a high level, qRAM allows an address register in superposition to access the corresponding stored data coherently, e.g., $\sum_i \alpha_i |i\rangle|0\rangle \mapsto \sum_i \alpha_i |i\rangle|\text{data}_i\rangle$. In our analysis, the stored data are indexed state-preparation data for the empirical $k$-mer distribution states, and we assume that such data can be queried coherently with polylogarithmic overhead. Such assumptions are common in high-level quantum algorithm analyses, but scalable qRAM is a strong architectural assumption and remains challenging in practice [12, 11, 3].

To apply these quantum search and estimation ideas to biological sequences, we use a distributional $k$-mer representation and a similarity measure that can be evaluated naturally within a quantum circuit. For each sequence, the empirical frequencies of its observed $k$-mers define a probability distribution over a global $k$-mer basis $\mathcal{K}$. Given two sequences $A$ and $B$ with empirical $k$-mer distributions $P_A$ and $P_B$, we compare them using the Bhattacharyya coefficient [5] $\mathrm{BC}(A, B) = \sum_{x \in \mathcal{K}} \sqrt{P_A(x) P_B(x)}$. This coefficient is large when two sequences assign similar weights to the same $k$-mers, so it measures the overlap between empirical $k$-mer distributions and captures alignment-free compositional similarity. Bhattacharyya-type distances have also appeared in previous alignment-free sequence-comparison work; for example, Zhang and Wang used the Bhattacharyya distance for alignment-free phylogenetic analysis of protein sequences [29].

Under square-root amplitude encoding, define the corresponding quantum states by $|\psi_A\rangle = \sum_{x \in \mathcal{K}} \sqrt{P_A(x)} |x\rangle$, $|\psi_B\rangle = \sum_{x \in \mathcal{K}} \sqrt{P_B(x)} |x\rangle$. Since the amplitudes are real and nonnegative, their inner product is $\langle \psi_A | \psi_B \rangle = \sum_{x \in \mathcal{K}} \sqrt{P_A(x) P_B(x)} = \mathrm{BC}(A, B)$. We refer to this inner product $\langle \psi_A | \psi_B \rangle$ as the state overlap between the two encoded sequence states. Thus the squared Bhattacharyya coefficient becomes the squared state overlap $|\langle \psi_A | \psi_B \rangle|^2$, which can be estimated by the SWAP test [8]. This connection is also related to quantum fingerprinting and quantum hashing, where short quantum states are used to represent classical inputs for comparison and communication tasks [2].

Moreover, quantum amplitude estimation (QAE) can estimate the probabilities produced by the SWAP test more efficiently than repeated sampling [7]: whereas naïve sampling requires $O(1/\varepsilon^2)$ repetitions to achieve additive error $\varepsilon$, QAE reduces this dependence to $O(1/\varepsilon)$ coherent queries.

Quantum nearest-neighbor and similarity-search algorithms have been studied previously in the quantum machine-learning literature. Wiebe et al. [28] developed coherent inner-product- and Euclidean-distance-estimation procedures for nearest-neighbor classification and clustering. Basheer et al. [4] considered a fidelity-based quantum $k$-nearest-neighbor algorithm and reduced the selection of the nearest states to quantum $k$-maxima finding. These works primarily address query-to-database search, in which a fixed test item is compared with a collection of training items. In contrast, our setting is an all-pairs closest-pair problem over a collection of classical biological sequences, with no distinguished query sequence.

Quantum search with imperfect predicates has also been studied in several related models. Høyer, Mosca, and de Wolf showed that bounded-error verifiers can be incorporated into quantum search without losing the quadratic query speedup, using a dedicated construction that interleaves amplitude amplification and error reduction [16]. Quek et al. [21] studied quantum minimum finding with an imprecise comparator, in which comparisons are guaran-

teed to be correct only when the compared values are separated by more than a prescribed resolution parameter. The latter model is particularly relevant to our QAE-based comparison circuit, because additive-error score estimates may not reliably distinguish pair scores that lie within the estimation precision.

In this work, we present a quantum framework for closest-pair search in biological sequence collections using the squared Bhattacharyya coefficient as the similarity score. Our central technical contribution is a coherent pairwise-score estimation circuit, together with a sparse prefix-tree construction for preparing the corresponding distribution states.

At the oracle-analysis level, we combine coherent indexed loading with Dürr–Høyer/BBHT maximum finding. Because a single bounded-error QAE execution does not by itself induce a fixed execution-wide comparison order, we state the main theorem under an explicit consistent $\varepsilon$-resolved coherent-comparison-oracle assumption. This assumption requires a fixed total order over the legal pairs that preserves every true score difference larger than $\varepsilon$ and admits an efficient clean coherent implementation. Under this assumption, the procedure achieves additive-$\varepsilon$ recovery using $O(N)$ expected comparison-oracle calls.

With idealized qRAM-style data access, the resulting conditional sequential gate complexity is $\widetilde{O}\left(\frac{NL}{\varepsilon}\right)$. For constant precision, this gives a quadratic reduction in the candidate-pair search component relative to the $O(N^2L)$ classical exhaustive baseline, conditional on the stated oracle and data-access assumptions.

We also implement a proof-of-concept version of the method in Q# [27]. The implementation uses finite-precision QAE, explicit multiplexed indexed loading, Grover-style search, and a Python controller that classically verifies measured candidates before updating the current candidate. The loading construction is related to probability-tree state-preparation methods [15, 13, 22]. The resulting workflow provides circuit-level validation of the implemented loading, estimation, marking, and search components.

Combining explicit multiplexed loading with the efficient consistent-order implementation assumed in the oracle-level analysis gives the conditional bound $\widetilde{O}\left(\frac{N^2L}{\varepsilon}\right)$, which, for constant precision, matches the $O(N^2L)$ classical exhaustive scaling up to polylogarithmic factors. The proof-of-concept workflow uses classically verified candidate updates and therefore differs from the fixed-order oracle procedure analyzed in the recovery theorem.

This is the first quantum framework specifically targeting alignment-free closest-pair search over biological sequence collections, and it provides both an oracle-level analysis and a circuit-level implementation platform for $k$-mer-distribution-based quantum sequence search.

## 2 Method

The method consists of four main components: (i) encode each sequence using the square roots of its empirical $k$-mer distribution; (ii) prepare pair-index superpositions and coherently load the corresponding sequence states; (iii) estimate pairwise similarities using a SWAP-test-based quantum amplitude estimation procedure; and (iv) apply Dürr–Høyer/BBHT maximum finding under a consistent $\varepsilon$-resolved coherent-comparison-oracle model.

### 2.1 Problem formulation and quantum state encoding

Let $S_1, S_2, \ldots, S_N$ be a collection of biological sequences over an alphabet $\Sigma$. Let $\mathcal{K} = \bigcup_{i=1}^{N} \{k\text{-mers appearing in } S_i\}$ be the set of distinct $k$-mers appearing in the input sequences, and write $M = |\mathcal{K}|$ for the size of this global observed $k$-mer basis.

For a sequence $X$ of length $L_X \geq k$, let $c_X(x)$ denote the number of occurrences of $x \in \mathcal{K}$ among the $L_X - k + 1$ consecutive length-$k$ windows of $X$. The empirical $k$-mer distribution of $X$ is

$$P_X(x) = \frac{c_X(x)}{L_X - k + 1}, \qquad x \in \mathcal{K}.$$

For two sequences $A$ and $B$, their Bhattacharyya coefficient is

$$\mathrm{BC}(A, B) = \sum_{x \in \mathcal{K}} \sqrt{P_A(x) P_B(x)}.$$

We use the squared Bhattacharyya coefficient as the similarity score, $F(A, B) = \mathrm{BC}(A, B)^2$. The closest-pair problem is therefore $(i^*, j^*) = \mathrm{argmax}_{1 \leq i < j \leq N} F(S_i, S_j)$.

We use a quantum register to mean a named collection of qubits that stores one part of the algorithmic state. To each sequence $X$, we associate the quantum state $|\psi_X\rangle = \sum_{x \in \mathcal{K}} \sqrt{P_X(x)}\,|x\rangle$. Since all amplitudes are real and nonnegative, the inner product between two such states is

$$\langle \psi_A | \psi_B \rangle = \sum_{x \in \mathcal{K}} \sqrt{P_A(x) P_B(x)} = \mathrm{BC}(A, B).$$

Therefore, $F(A, B) = |\langle \psi_A | \psi_B \rangle|^2$. Thus, closest-pair search under $F$ is equivalent to finding the encoded sequence pair with maximum squared state overlap.

## 2.2 Coherent indexed state loading

The maximum-finding procedure requires coherent access to the encoded sequence states. We denote the corresponding indexed-loading operation by $U_{\mathrm{load}} : |i\rangle|0\rangle \longmapsto |i\rangle|\psi_{S_i}\rangle$, and write $C_{\mathrm{load}}$ for its sequential gate cost.

The operation $U_{\mathrm{load}}$, its inverse $U_{\mathrm{load}}^\dagger$, and their controlled versions must act coherently when the index register is in superposition. Below, we give an explicit multiplexed construction realizing this operation. For the improved qRAM-based complexity bound, we separately introduce an idealized indexed-access assumption.

Given an index register in a superposition $\sum_i \alpha_i |i\rangle$, linearity gives $\sum_i \alpha_i |i\rangle|0\rangle \longmapsto \sum_i \alpha_i |i\rangle|\psi_{S_i}\rangle$. For a candidate pair $(i, j)$, the loading operation is applied twice:

$$|i, j\rangle|0\rangle|0\rangle \longmapsto |i, j\rangle|\psi_{S_i}\rangle|\psi_{S_j}\rangle.$$

### 2.2.1 Pair-index search state

Our method searches over candidate sequence pairs by representing the pair indices in quantum registers. We prepare a uniform superposition over the ordered index space $\mathcal{R} = \{1, \ldots, N\} \times \{1, \ldots, N\}$. This gives the pair-index state

$$|\Psi_{\mathcal{R}}\rangle = \left( \frac{1}{\sqrt{N}} \sum_{i=1}^{N} |i\rangle \right) \otimes \left( \frac{1}{\sqrt{N}} \sum_{j=1}^{N} |j\rangle \right) = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{N} |i, j\rangle.$$

### 2.2.2 Sparse prefix-tree preparation of a single k-mer distribution

For each sequence $X$, the implementation prepares the state $|\psi_X\rangle = \sum_{x \in \mathcal{K}} \sqrt{P_X(x)}\,|x\rangle$ from its empirical $k$-mer distribution $P_X$ using a sparse binary prefix-tree construction.

Let $d = \lceil \log_2 M \rceil$, and identify each $k$-mer basis element in $\mathcal{K}$ with a $d$-bit label, padding the basis to dimension $2^d$ if necessary. The preparation circuit is defined over the binary prefix tree of these $d$-bit labels.

For a prefix $u \in \{0,1\}^\ell$, define the probability mass $\mu_X(u) = \sum_{x:\, x \text{ has prefix } u} P_X(x)$. Thus $\mu_X(u)$ is the total probability mass of all $k$-mers whose binary labels begin with $u$. At an internal prefix node $u$, the mass splits between its two children $u_0$ and $u_1$. Conditional on the register already having prefix $u$, the next qubit should be prepared with probabilities

$$\Pr(0 \mid u) = \frac{\mu_X(u_0)}{\mu_X(u)}, \qquad \Pr(1 \mid u) = \frac{\mu_X(u_1)}{\mu_X(u)}.$$

Using the convention $R_y(\theta)|0\rangle = \cos(\theta/2)|0\rangle + \sin(\theta/2)|1\rangle$, this conditional split is implemented by a prefix-controlled $R_y$ rotation on the next qubit with angle $\theta_u = 2 \arcsin \sqrt{\frac{\mu_X(u_1)}{\mu_X(u)}}$. After this rotation, the branch $u_0$ receives amplitude factor $\sqrt{\frac{\mu_X(u_0)}{\mu_X(u)}}$, and the branch $u_1$ receives amplitude factor $\sqrt{\frac{\mu_X(u_1)}{\mu_X(u)}}$.

Applying these rotations from the root to the leaves prepares the target state. For a leaf $x = x_1 x_2 \cdots x_d$, the amplitude accumulated along its root-to-leaf path is

$$\prod_{\ell=0}^{d-1} \sqrt{\frac{\mu_X(x_1 \cdots x_{\ell+1})}{\mu_X(x_1 \cdots x_\ell)}} = \sqrt{\mu_X(x)} = \sqrt{P_X(x)},$$

where the product telescopes and $\mu_X(\emptyset) = 1$. Therefore the prepared state has exactly the desired square-root amplitudes.

▶ **Assumption 1** (Idealized qRAM-style indexed loading)**.** *For a sequence index $i$ and a position in the sparse state-preparation schedule of $S_i$, the corresponding rotation data can be queried coherently with overhead*

$$\eta_{\text{idx}} = \text{polylog}(N, L, M).$$

*Under this access model, the indexed-loading operation $U_{\text{load}}$, its inverse, and their controlled versions can be implemented with sequential gate cost*

$$C_{\text{load}}^{\text{qRAM}} = O\big(L \log M \, (\eta_{\text{idx}} + \log M)\big) = \widetilde{O}(L).$$

### 2.2.3 Explicit indexed loading from sparse state preparations

We now give an explicit non-qRAM construction of the coherent indexed-loading operation $U_{\text{load}}$ used in the Q# implementation. This construction does not require Assumption 1.

Let $P_t$ denote the sparse prefix-tree preparation circuit for sequence $S_t$. The indexed loader iterates over $t \in \{1, \ldots, N\}$, reversibly checks whether the index register equals $t$, and conditionally applies $P_t$ to the sequence-state register. Thus, $|t\rangle|0\rangle \longmapsto |t\rangle|\psi_{S_t}\rangle$ for every computational-basis index $t$, and the same operation acts coherently on superpositions by linearity.

For a candidate pair, the construction is applied independently to the two sequence indices: $|i,j\rangle|0\rangle|0\rangle \longmapsto |i,j\rangle|\psi_{S_i}\rangle|\psi_{S_j}\rangle$.

Because the construction explicitly multiplexes over all stored preparation circuits, its indexed-loading cost depends on the total preparation-schedule size over all $N$ sequences. This cost is derived in Section 3.

## 2.3 Coherent pairwise-score estimation

After indexed loading, each pair-index basis state $|i,j\rangle$ is associated with the sequence states $|\psi_{S_i}\rangle$ and $|\psi_{S_j}\rangle$. By the square-root amplitude encoding, $\langle\psi_{S_i} \mid \psi_{S_j}\rangle = \mathrm{BC}(S_i,S_j)$, and hence $F(S_i,S_j) = \mathrm{BC}(S_i,S_j)^2 = \left|\langle\psi_{S_i} \mid \psi_{S_j}\rangle\right|^2$.

To access this score coherently, we apply a SWAP test to the two loaded sequence-state registers. For arbitrary pure states $|\phi\rangle$ and $|\varphi\rangle$, the probability of measuring the SWAP-test ancilla in state $|0\rangle$ is $p = \frac{1+|\langle\phi|\varphi\rangle|^2}{2}$. Therefore, for pair $(i,j)$, $p_{ij} = \frac{1+\left|\langle\psi_{S_i}|\psi_{S_j}\rangle\right|^2}{2} = \frac{1+F(S_i,S_j)}{2}$, or equivalently, $F(S_i,S_j) = 2p_{ij} - 1$.

Let $A_{ij}$ denote the coherent circuit that loads the two sequence states and applies the SWAP test. It prepares a state of the form $A_{ij}|0\rangle = \sqrt{p_{ij}}\,|\mathrm{good}_{ij}\rangle + \sqrt{1-p_{ij}}\,|\mathrm{bad}_{ij}\rangle$, where the good subspace is defined by the SWAP-test ancilla being in state $|0\rangle$.

Over a superposition of pair indices, the corresponding controlled operation is $A = \sum_{i,j} |i,j\rangle\langle i,j| \otimes A_{ij}$.

Let $\varepsilon \in (0,1)$ denote the desired final additive accuracy of the returned pair score, and define the internal score-estimation accuracy $\xi = \frac{\varepsilon}{2}$. Quantum amplitude estimation is used to estimate $p_{ij}$ to additive error at most $\xi/2$. A measurement of the resulting estimate register gives a score estimate $\widetilde{F}_{ij} = 2\widetilde{p}_{ij} - 1$ satisfying $\left|\widetilde{F}_{ij} - F(S_i,S_j)\right| \le \xi$ with constant success probability.

Obtaining this precision requires $O\left(\frac{1}{\xi}\right) = O\left(\frac{1}{\varepsilon}\right)$ coherent applications of $A$, $A^\dagger$, and the associated reflections.

A single bounded-error QAE execution does not by itself define the execution-wide deterministic comparison order required by the standard Dürr–Høyer analysis. In particular, repeated coherent executions need not induce the same ordering among candidate pairs whose scores are close, and the resulting marking operation is not automatically a clean Boolean phase oracle on the pair-index register.

We therefore separate the coherent score-estimation construction from the oracle-level maximum-finding analysis. The QAE construction shows that additive score estimation uses $O\left(\frac{1}{\varepsilon}\right)$ applications of the overlap-estimation block and motivates the cost assumed for one coherent comparison. The main theorem, however, is stated under the additional efficient-consistent-order assumption of Assumption 3. That assumption requires one fixed deterministic comparison order, a clean coherent implementation of that order, and the stated implementation cost.

Establishing Assumption 3 directly from the bounded-error QAE circuit would require an additional construction ensuring execution-wide consistency and clean uncomputation. Such a construction is outside the scope of the present work and is not implemented in the Q# proof of concept. An alternative route would be to analyze the QAE-based comparison procedure within a margin-based imprecise-comparator model and apply robust quantum minimum- or maximum-finding techniques. Such an analysis may introduce an additional dependence on the number of candidate scores lying within the comparison margin and is likewise left for future work.

## 2.4 Consistent comparison oracle and quantum maximum finding

Within the ordered pair-index space $\mathcal{R}$ defined above, the legal closest-pair candidates are the canonically oriented unordered pairs $\mathcal{P} = \{(i,j) : 1 \le i < j \le N\}$, $|\mathcal{P}| = \binom{N}{2}$. The legality condition $i < j$ is checked reversibly inside the marking circuit, thereby excluding self-pairs and reversed duplicate pairs.

For a current classical candidate $y \in \mathcal{P}$, the maximum-finding procedure requires a coherent comparison operation that marks pairs ranked above $y$. We formalize this requirement in the following definition.

▶ **Definition 2** (Consistent $\varepsilon$-resolved comparison oracle). *A family of coherent phase oracles* $\left\{O_y^{(\varepsilon)} : y \in \mathcal{P}\right\}$ *is a consistent $\varepsilon$-resolved comparison-oracle family if there exists a strict total order $\succ_\varepsilon$ on $\mathcal{P}$ such that $O_y^{(\varepsilon)}|x\rangle = (-1)^{[x \succ_\varepsilon y]}|x\rangle$ for every $x, y \in \mathcal{P}$, with all score-estimation, loading, and comparison work registers returned to their initial states.*

*The order must preserve every score difference larger than $\varepsilon$: $F(x) > F(y) + \varepsilon \implies x \succ_\varepsilon y$. Pairs whose scores differ by at most $\varepsilon$ may be ordered arbitrarily, provided that the same total order is used consistently throughout the execution.*

▶ **Assumption 3** (Efficient coherent implementation of a consistent $\varepsilon$-resolved order). *There exists a fixed deterministic ordering key $K_\varepsilon : \mathcal{P} \longrightarrow \{0,1\}^b$ whose lexicographic order induces a strict total order $\succ_\varepsilon$ on the legal candidate pairs. The same key and the same tie-breaking rule are used throughout the entire execution.*

*Without loss of generality, the key is injective and has $b = O(\log N + \log(1/\varepsilon))$ bits, including a deterministic pair-index tie-breaking component. We additionally assume that $U_{K_\varepsilon}$ and the reversible comparison circuit use $q_K = \mathrm{polylog}(N, L, M, 1/\varepsilon)$ work qubits.*

*The induced order preserves every score difference larger than $\varepsilon$:*

$$F(x) > F(y) + \varepsilon \quad \Longrightarrow \quad K_\varepsilon(x) >_{\mathrm{lex}} K_\varepsilon(y),$$

*and hence $x \succ_\varepsilon y$.*

*We assume that the key can be evaluated coherently by a reversible circuit $U_{K_\varepsilon} : |x\rangle|0\rangle \longmapsto |x\rangle|K_\varepsilon(x)\rangle$, and that $U_{K_\varepsilon}$, its inverse, and their controlled versions have sequential gate cost $\widetilde{O}\left(\frac{C_{\mathrm{load}} + \log M}{\varepsilon}\right)$. Using a reversible lexicographic comparison followed by uncomputation, this circuit implements the clean phase oracle $O_y^{(\varepsilon)}|x\rangle = (-1)^{[x \succ_\varepsilon y]}|x\rangle$, with all key, loading, score-estimation, and comparison work registers returned to their initial states.*

*This assumption includes both the execution-wide consistency of the comparison order and the existence of an efficient clean coherent implementation of that order. It is not implied by the bounded-error guarantee of a single QAE execution. The present Q# proof-of-concept implementation does not construct $U_{K_\varepsilon}$ or establish this assumption.*

Under Assumption 3, the Dürr–Høyer procedure is applied to the fixed total order $\succ_\varepsilon$. For the current classical candidate $y$, the comparison oracle marks precisely the legal pairs $x$ satisfying $x \succ_\varepsilon y$. The number of marked pairs is not known in advance, so each search step uses a BBHT-style randomized Grover-iteration schedule [6].

Whenever the oracle-level search returns a marked pair $x$, the current candidate is updated to $x$. Thus the candidate rank improves strictly according to the fixed order $\succ_\varepsilon$, as required by the standard Dürr–Høyer analysis.

The Q# proof-of-concept workflow described in Section 4 instead evaluates the measured pair classically and accepts it only when its true score improves. That hybrid verification rule is useful experimentally, but it defines a different candidate-update procedure and is not covered by the oracle-level recovery guarantee below.

▶ **Theorem 4** (Main result). *Under the efficient coherent comparison-order implementation assumed in Assumption 3, let $S_1, \ldots, S_N$ be biological sequences of length at most $L$, and define $F^\star = \max_{1 \le i < j \le N} F(S_i, S_j)$.*

*The Dürr–Høyer maximum-finding procedure returns, with probability at least* $2/3$*, a legal pair* $\widehat{y} = (\widehat{i}, \widehat{j})$ *satisfying* $F(S_{\widehat{i}}, S_{\widehat{j}}) \geq F^\star - \varepsilon$.

*Under the implementation-cost component of Assumption 3, the expected conditional sequential gate complexity is* $\widetilde{O}\left(\frac{N(C_{\mathrm{load}}+\log M)}{\varepsilon}\right)$.

*Combining Assumption 3 with the idealized qRAM-style loading Assumption 1 gives the conditional bound* $\widetilde{O}\left(\frac{NL}{\varepsilon}\right)$. *Combining the same comparison-order implementation assumption with the explicit multiplexed loading cost gives* $\widetilde{O}\left(\frac{N^2 L}{\varepsilon}\right)$.

**Proof.** Because each successful update replaces the current candidate by a pair that is strictly larger under $\succ_\varepsilon$, the candidate-update process is exactly the Dürr–Høyer procedure applied to this fixed total order. By Assumption 3, the comparison operations used throughout the execution are induced by one fixed strict total order $\succ_\varepsilon$. The standard Dürr–Høyer maximum-finding analysis therefore applies to this order. Since $|\mathcal{P}| = \binom{N}{2} = \Theta(N^2)$, the procedure finds a maximal element $\widehat{y}$ of $\succ_\varepsilon$ with constant success probability using $O\left(\sqrt{|\mathcal{P}|}\right) = O(N)$ comparison-oracle applications in expectation. A constant number of independent repetitions raises the success probability to at least $2/3$.

Let $x^\star$ be a true optimal pair. If $F(\widehat{y}) < F^\star - \varepsilon$, then $F(x^\star) > F(\widehat{y}) + \varepsilon$. Definition 2 would therefore imply $x^\star \succ_\varepsilon \widehat{y}$, contradicting the maximality of $\widehat{y}$. Hence $F(\widehat{y}) \geq F^\star - \varepsilon$.

By Assumption 3, one comparison-oracle application costs

$$O\left(\frac{C_{\mathrm{load}} + \log M}{\varepsilon}\right)$$

up to polylogarithmic overheads. Multiplying by the $O(N)$ expected oracle applications gives $\widetilde{O}\left(\frac{N(C_{\mathrm{load}}+\log M)}{\varepsilon}\right)$. Substituting the two indexed-loading costs gives the stated qRAM and explicit-loading bounds. ◀

### 2.4.1 Exact recovery under a score-gap condition

If a strictly suboptimal score exists, let $F^{(2)}$ denote the largest score strictly below $F^\star$. If every legal pair is optimal, exact recovery is immediate.

▶ **Assumption 5** (Optimal-score gap)**.** *The optimal and strictly suboptimal scores satisfy* $F^\star - F^{(2)} > \varepsilon$.

▶ **Corollary 6** (Exact recovery)**.** *Under the assumptions of Theorem 4 and Assumption 5, the returned pair is exactly optimal whenever the maximum-finding procedure succeeds.*

**Proof.** Let $x^\star$ be any optimal pair and let $y$ be strictly suboptimal. Then $F(x^\star) - F(y) \geq F^\star - F^{(2)} > \varepsilon$. Definition 2 therefore gives $x^\star \succ_\varepsilon y$. No strictly suboptimal pair can be maximal in $\succ_\varepsilon$, so every maximal element returned by the maximum-finding procedure is optimal. ◀

## 3 Complexity analysis and comparison

The complexity bounds in this section are conditional on Assumption 3. We first express the quantum cost in terms of the abstract indexed-loading cost $C_{\mathrm{load}}$, and then instantiate it under the idealized qRAM-style and explicit multiplexed loading models. We also derive the classical preprocessing cost and logical qubit count.

Throughout this section, the alphabet $\Sigma$ is treated as fixed, while the $k$-mer length $k$, and hence the observed basis size $M \leq |\Sigma|^k$, may vary with the input. Thus extracting the overlapping $k$-mers of a length-$L$ sequence takes $O(L)$ word operations. We retain the dependence on the observed $k$-mer basis size $M = |\mathcal{K}|$, with $M \leq |\Sigma|^k$, because $M$ determines the dimension of the encoded states and the size of the $k$-mer basis register.

Let $s_i$ be the number of distinct observed $k$-mers in sequence $S_i$. If all sequences have length at most $L$, then $s_i \leq L_i - k + 1 \leq L - k + 1 = O(L)$.

## 3.1 Classical exhaustive baseline

An exact classical closest-pair search evaluates the similarity score for all $\binom{N}{2} = \Theta(N^2)$ candidate pairs. Using a hashmap for the $k$-mer counts, the Bhattacharyya coefficient between two sequences can be computed by iterating over the nonzero $k$-mer counts of one sequence and looking up the corresponding counts in the other. Thus one pairwise comparison costs $O(L)$ time in the worst case, and the classical exhaustive baseline is $T_{\text{classical}} = O(N^2 L)$.

## 3.2 Classical preprocessing

Before running the quantum circuit, we compute $k$-mer counts for each input sequence, construct the global observed $k$-mer basis $\mathcal{K}$, normalize each sequence into an empirical distribution, and construct the square-root amplitude representation. Extracting $k$-mer counts takes $O(NL)$ time.

In addition, the implementation constructs a sparse state-preparation schedule for each sequence. By a state-preparation schedule, we mean the classical list of prefix-controlled $R_y$ rotations generated by the sparse prefix-tree construction. Let $R_i$ denote the number of rotations in the schedule for sequence $S_i$, and let $R_{\text{tot}} = \sum_{i=1}^{N} R_i$ be the total schedule size over all sequences. The schedule construction uses cached prefix probability masses in the binary preparation tree. For sequence $S_i$, the schedule size satisfies $R_i = O(s_i \log M)$, and the schedule can be constructed in $O(s_i \log M)$ time. Therefore, $R_{\text{tot}} = O\left(\sum_{i=1}^{N} s_i \log M\right)$. Since $s_i = O(L)$, we have

$$R_{\text{tot}} = O(NL \log M). \tag{1}$$

Thus the preprocessing time is $O(NL \log M)$.

## 3.3 Indexed loading cost

Let $C_{\text{load}}$ denote the gate cost of coherently loading one indexed sequence state, $U_{\text{load}} : |i\rangle|0\rangle \mapsto |i\rangle|\psi_{S_i}\rangle$. We keep $C_{\text{load}}$ abstract first, and later substitute the cost under a qRAM-style model and under our concrete implementation.

For a queried pair $(i, j)$, the SWAP-test block applies $U_{\text{load}}$ twice, once for $i$ and once for $j$, and then performs a controlled-SWAP between the two sequence-state registers. The sequence-state register has $d = \lceil \log_2 M \rceil$ qubits, so the controlled-SWAP part costs $O(\log M)$. Therefore one coherent overlap-estimation block costs $O(C_{\text{load}} + \log M)$, where the factor of two from loading two states is absorbed into the big-$O$ notation.

## 3.4 Cost of one comparison-oracle application

The following oracle cost is an explicit component of Assumption 3.

The coherent score estimator uses the internal precision $\xi = \frac{\varepsilon}{2}$. Estimating the SWAP-test probability to additive error $\xi/2$ requires $O\left(\frac{1}{\xi}\right) = O\left(\frac{1}{\varepsilon}\right)$ coherent applications of the overlap-estimation block and its inverse. Thus the score-estimation component of one threshold-comparison query costs $O\left(\frac{C_{\text{load}}+\log M}{\varepsilon}\right)$.

The threshold-search circuit also prepares and reflects about the ordered pair-index search state $|\Psi_{\mathcal{R}}\rangle = |u_N\rangle \otimes |u_N\rangle$, $|u_N\rangle = \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle$. Here the computational-basis label $i \in \{0, \ldots, N-1\}$ corresponds to sequence $S_{i+1}$.

We prepare $|u_N\rangle$ exactly by a recursive binary decomposition. Let $m = \lceil \log_2 N \rceil$. If $N = 2^m$, then $|u_N\rangle = H^{\otimes m}|0^m\rangle$. Otherwise, write $N = 2^{m-1} + r$, $1 \le r < 2^{m-1}$. The most significant qubit is first prepared as $\sqrt{\frac{2^{m-1}}{N}}\,|0\rangle + \sqrt{\frac{r}{N}}\,|1\rangle$ by applying $R_y\left(2 \arcsin \sqrt{\frac{r}{N}}\right)$. Conditioned on the most significant qubit being 0, Hadamard gates are applied to the remaining $m-1$ qubits. Conditioned on it being 1, the same construction is applied recursively to prepare $|u_r\rangle$ on the remaining qubits. The resulting state is

$$\sqrt{\frac{2^{m-1}}{N}}\,|0\rangle \left( \frac{1}{\sqrt{2^{m-1}}} \sum_{x=0}^{2^{m-1}-1} |x\rangle \right) + \sqrt{\frac{r}{N}}\,|1\rangle \left( \frac{1}{\sqrt{r}} \sum_{x=0}^{r-1} |x\rangle \right) = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle.$$

At each of the $m = O(\log N)$ recursion levels, the construction uses at most $O(m)$ controlled one- and two-qubit operations. Therefore, using standard controlled-gate decompositions, one preparation of $|u_N\rangle$ requires $O(\log^2 N)$ elementary gates. Preparing the two-register product state $|\Psi_{\mathcal{R}}\rangle$ has the same asymptotic cost.

Let $U_N|0^m\rangle = |u_N\rangle$. Reflection about the pair-index search state is implemented as

$$(U_N \otimes U_N)\left(2|0^{2m}\rangle\langle 0^{2m}| - I\right)(U_N^\dagger \otimes U_N^\dagger).$$

The middle operation is a multi-controlled phase on $2m = O(\log N)$ qubits and has polynomial-in-$\log N$ gate cost under standard decompositions. Thus both preparation of and reflection about $|\Psi_{\mathcal{R}}\rangle$ cost $O(\log^2 N)$ gates, up to the chosen ancilla–gate tradeoff. The reversible validity test $i < j$ can be implemented by a standard $m$-bit comparator using $O(\log N)$ gates and work qubits.

Hence, under Assumption 3, one comparison-oracle application has cost $C_{\text{query}} = O\left(\frac{C_{\text{load}}+\log M}{\varepsilon}\right)$ up to logarithmic overheads from pair-index preparation, reflection, and reversible comparisons. Accordingly, throughout the oracle-level complexity analysis we assume $C_{\text{query}} = \widetilde{O}\left(\frac{C_{\text{load}}+\log M}{\varepsilon}\right)$. This bound is conditional on the efficient coherent implementation postulated in Assumption 3.

## 3.5 Number of comparison-oracle queries

By Theorem 4, maximum finding over $|\mathcal{P}| = \Theta(N^2)$ legal pairs uses $O(N)$ expected comparison-oracle applications. Combining this with $C_{\text{query}} = \widetilde{O}\left(\frac{C_{\text{load}}+\log M}{\varepsilon}\right)$ gives

$$T_{\text{abstract}} = \widetilde{O}\left(\frac{N(C_{\text{load}} + \log M)}{\varepsilon}\right).$$

## 3.6 Number of qubits

Let $m = \lceil \log_2 N \rceil, d = \lceil \log_2 M \rceil, r = \lceil \log_2(1/\varepsilon) \rceil$ be the sequence-index register width, the $k$-mer basis register width, and the QAE evaluation-register width, respectively. The circuit uses two index registers, two sequence-state registers, one QAE evaluation register, and a

constant number of ancilla qubits for the SWAP-test and marking logic. Ignoring temporary work qubits used to decompose arithmetic and multi-controlled gates, the logical qubit count is $Q_{\text{logical}} = 2m + 2d + r + O(1)$. Including reversible comparison and gate-decomposition workspace, the qubit count is $Q = O(\log N + \log M + \log(1/\varepsilon))$. The sparse state-preparation schedules are classical data supplied to the circuit, and are not counted as qubits.

## 3.7 Total complexity under idealized qRAM-style data access

Under Assumption 1, the indexed-loading operation has sequential gate cost $C_{\text{load}}^{\text{qRAM}} = O\big(L \log M \left(\eta_{\text{idx}} + \log M\right)\big) = \widetilde{O}(L)$.

Substituting this loading cost into the comparison-oracle cost bound gives $C_{\text{query}}^{\text{qRAM}} = O\left(\frac{C_{\text{load}}^{\text{qRAM}} + \log M}{\varepsilon}\right)$. Hence, $C_{\text{query}}^{\text{qRAM}} = O\left(\frac{L \log M \left(\eta_{\text{idx}} + \log M\right)}{\varepsilon}\right) = \widetilde{O}\left(\frac{L}{\varepsilon}\right)$.

Under Assumption 3, the Dürr–Høyer procedure uses $O(N)$ expected comparison-oracle applications over the $\Theta(N^2)$ legal candidate pairs. Therefore, the total sequential gate complexity is

$$T_{\text{qRAM}} = O\big(N C_{\text{query}}^{\text{qRAM}}\big) = O\left(\frac{NL \log M \left(\eta_{\text{idx}} + \log M\right)}{\varepsilon}\right) = \widetilde{O}\left(\frac{NL}{\varepsilon}\right).$$

For constant precision, the leading dependence on the number of sequences is linear rather than quadratic. Compared with the classical exhaustive baseline $T_{\text{classical}} = O(N^2 L)$, this reduces the number of comparison-oracle applications from $\Theta(N^2)$ classically to $O(N)$ quantumly, conditional on the availability of the assumed comparison oracle.

## 3.8 Total complexity under explicit non-qRAM loading

For the explicit multiplexed construction of Section 2.2, one indexed loading operation applies the stored preparation schedules conditionally over all sequence indices. Its cost is therefore $C_{\text{load}}^{\text{impl}} = O(R_{\text{tot}}(\log M + \log N))$.

Using Eq. (1), this becomes $C_{\text{load}}^{\text{impl}} = O(NL \log M(\log M + \log N))$.

Substituting this loading cost into the comparison-oracle cost bound gives $C_{\text{query}}^{\text{impl}} = O\left(\frac{NL \log M(\log M + \log N)}{\varepsilon}\right)$.

Under Assumption 3, the Dürr–Høyer procedure uses $O(N)$ expected comparison-oracle applications. Therefore, the total sequential gate complexity is

$$T_{\text{impl}} = O\big(N C_{\text{query}}^{\text{impl}}\big) = O\left(\frac{N^2 L \log M(\log M + \log N)}{\varepsilon}\right) = \widetilde{O}\left(\frac{N^2 L}{\varepsilon}\right).$$

For constant precision, this matches the classical exhaustive baseline $T_{\text{classical}} = O(N^2 L)$ up to polylogarithmic factors. Thus the explicit non-qRAM construction makes the coherent indexed-loading cost explicit, but does not provide an end-to-end asymptotic speedup over exhaustive classical comparison under the stated comparison-oracle model.

# 4 Implementation and simulation

We implemented a proof-of-concept version of the method in Q# with a Python driver to validate the circuit-level construction. The source code, experiment scripts, and data-processing utilities are available at `https://github.com/Kingsford-Group/tieria`.

## 4.1 Implementation details

The implementation realizes the circuit-level components described in Section 2. A Python driver performs the classical preprocessing: it extracts observed $k$-mers, constructs the global basis $\mathcal{K}$, computes empirical $k$-mer distributions, and generates the sparse prefix-tree preparation schedules used by the Q# circuit.

The Q# component implements pair-index state preparation, multiplexed indexed loading from sparse schedules, SWAP-test-based QAE, threshold marking, and Grover-style search. The indexed loading operation uses the explicit non-qRAM construction analyzed in Section 3; that is, the circuit multiplexes over the stored sequence schedules rather than assuming qRAM access.

The outer candidate-update loop is controlled by the Python driver. At each iteration, the driver stores the current candidate pair and computes its threshold score from the preprocessed amplitude vectors. The Q# circuit then performs a finite-precision BBHT-style search, and the driver accepts a measured pair only when its classically evaluated score is strictly larger than that of the current candidate.

This hybrid update rule is not the oracle-order update rule used in Theorem 4. The oracle-level theorem updates according to a fixed total order $\succ_\varepsilon$, whereas the proof-of-concept implementation updates according to classically verified true-score improvement.

The implementation therefore exercises the explicit loading, SWAP-test-based estimation, finite-precision marking, and Grover-style search components, but it neither constructs the deterministic key $K_\varepsilon$ nor establishes Assumption 3. The simulation results are reported only as small-scale empirical validation of the implemented workflow and do not establish the theorem's additive-$\varepsilon$ or exact-recovery guarantee.

## 4.2 Simulation setup

We evaluate the proof-of-concept implementation in two complementary ways. First, we use the Q# simulator to test whether the search procedure can recover an exact closest pair found by classical exhaustive enumeration on small random DNA instances. Second, we estimate logical circuit resources for the generated coherent search circuits.

We generated random DNA sequence collections over the alphabet $\Sigma = \{A, C, G, T\}$. The experiments vary the $k$-mer length $k$, sequence length $L$, and number of sequences $N$. We use one-parameter sweeps designed to isolate the effects of $k$, $L$, and $N$:

$$
\begin{aligned}
&k\text{-sweep:} && (k, L, N) \in \{(2, 12, 3), (3, 12, 3), (4, 12, 3)\},\\
&L\text{-sweep:} && (k, L, N) \in \{(2, 5, 3), (2, 8, 3), (2, 12, 3), (2, 16, 3)\},\\
&N\text{-sweep:} && (k, L, N) \in \{(2, 8, 3), (2, 8, 4), (2, 8, 5), (2, 8, 6), (2, 8, 7), (2, 8, 8)\}.
\end{aligned}
$$

We also include the additional settings $(k, L, N) \in \{(3, 16, 3), (4, 16, 3)\}$ to test larger $k$-mer representations at the longer sequence length $L = 16$.

For the Q# simulator runs, we use $r = 3$ QAE precision qubits, a BBHT growth factor of 1.2, and a maximum cumulative Grover-iteration budget of 120. With 3 QAE evaluation qubits, the nominal resolution of the QAE output is $2^{-r} = 1/8$. Thus, these experiments use only coarse score estimation. The initial candidate pair is chosen in a nonoptimal mode whenever possible, so that the threshold-search procedure must improve the current candidate. For each random instance, we compute the exact optimal pair classically by exhaustive enumeration over all legal pairs. A run is counted as successful if the first optimal pair observed by the Q# search is one of the exact optimal pairs, allowing for ties.

For correctness, we report the success rate over random trials. For search-level behavior, we report the number of Q# circuit calls and the cumulative number of Grover iterations used before the first exact optimal pair is observed. These quantities characterize the time-to-first-optimal behavior of the Dürr–Høyer/BBHT threshold-search procedure under the configured search budget. We do not report simulator wall-clock time, since it primarily reflects classical state-vector simulation overhead rather than quantum hardware runtime.

## 4.3 Q# resource estimation

We estimate logical circuit resources using `qsharp.logical_counts`. The reported quantities include logical qubits, logical depth, rotation counts, and $T$-gate-related counts returned by the Q# toolchain. These estimates are obtained from coherent circuit entry points used for resource counting, rather than from simulator wall-clock time.

To relate these resource counts to the structure of the generated instances, we also record several quantities from the classical preprocessing. These include the observed $k$-mer basis size $M = |\mathcal{K}|$, the average and maximum nonzero $k$-mer support size, and the total sparse state-preparation schedule size $R_{\mathrm{tot}} = \sum_{i=1}^{N} R_i$, where $R_i$ is the number of sparse state-preparation schedule entries used for sequence $S_i$.

## 4.4 Proof-of-concept circuit validation

Table 1 summarizes the proof-of-concept results for the implemented Python–Q# workflow. Across all 13 parameter settings and 65 random trials, the workflow observed an exact optimal pair before the configured Grover-iteration cap, giving an empirical success rate of 100% on the tested synthetic instances. Exact optimality was determined independently by classical exhaustive enumeration.

These results demonstrate that the implemented circuit subroutines – indexed state loading, SWAP-test-based QAE, threshold marking, and Grover-style search – can be integrated successfully within the hybrid workflow.

For descriptive purposes, the $N$-sweep at fixed $k = 2$ and $L = 8$ shows how the measured search-level quantities change over the small tested range. The number of legal pairs increases from $\binom{3}{2} = 3$ at $N = 3$ to $\binom{8}{2} = 28$ at $N = 8$. Over the same range, the average number of Q# circuit calls before first observing an exact optimal pair increases from 3.4 to 7.6, while the average cumulative number of Grover iterations increases from 3.6 to 13.0.

Because the experiments use only $N \leq 8$, synthetic random-DNA inputs, and five trials per setting, these measurements are intended only as circuit-level proof-of-concept validation.

## 4.5 Logical resource estimates

Table 2 reports the logical resource estimates obtained using `qsharp.logical_counts`. The table reports estimates for circuits with $r = 3$ QAE evaluation qubits and one Grover iteration. Each row is averaged over three random trials. All logical-count runs completed successfully.

The logical resource estimates show that circuit resources grow with the structure of the generated state-preparation schedules. In the $N$-sweep with $k = 2$ and $L = 8$, $R_{\mathrm{tot}}$ increases from 22.3 at $N = 3$ to 61.7 at $N = 8$. Over the same sweep, the mean logical depth increases from 1264.3 to 4201.3, and the reported rotation count increases from 1395.0 to 4389.0. This is consistent with the concrete implementation, which performs explicit sequence-indexed loading over the stored sequence states rather than assuming qRAM.

The dependence on $k$ and $L$ is more instance-dependent. Increasing $L$ at fixed $k = 2, N = 3$ increases the support size and sparse schedule size, and the logical depth grows from 668.0 at $L = 5$ to 3398.3 at $L = 16$. For the $k$-sweep at fixed $L = 12, N = 3$, the observed basis

**Table 1** Proof-of-concept results for the Python–Q# workflow on selected synthetic random-DNA instances. Each row reports averages over five random trials. "Classical pairs" is the number of legal pairs $\binom{N}{2}$ evaluated by exhaustive classical closest-pair search. Q# calls and Grover iterations are measured until the workflow first observes an exact optimal pair, with optimality verified independently by classical exhaustive enumeration. Here $M = |\mathcal{K}|$ is the observed $k$-mer basis size and $R_{\text{tot}} = \sum_i R_i$ is the total sparse state-preparation schedule size over all sequences.

| $k$ | $L$ | $N$ | Classical pairs | Q# calls | Grover iters | Success | $M$ | $R_{\text{tot}}$ |
|---|---|---|---|---|---|---|---|---|
| 2 | 5 | 3 | 3 | 1.6 | 1.6 | 5/5 | 8.2 | 11.6 |
| 2 | 8 | 3 | 3 | 3.4 | 3.6 | 5/5 | 11.8 | 21.4 |
| 2 | 8 | 4 | 6 | 5.8 | 7.2 | 5/5 | 14.0 | 29.8 |
| 2 | 8 | 5 | 10 | 4.2 | 4.2 | 5/5 | 15.0 | 38.6 |
| 2 | 8 | 6 | 15 | 5.2 | 5.6 | 5/5 | 15.2 | 45.6 |
| 2 | 8 | 7 | 21 | 5.6 | 6.4 | 5/5 | 15.4 | 54.4 |
| 2 | 8 | 8 | 28 | 7.6 | 13.0 | 5/5 | 15.4 | 63.0 |
| 2 | 12 | 3 | 3 | 2.4 | 2.4 | 5/5 | 13.2 | 26.6 |
| 2 | 16 | 3 | 3 | 2.6 | 2.6 | 5/5 | 15.0 | 32.8 |
| 3 | 12 | 3 | 3 | 6.4 | 7.8 | 5/5 | 24.8 | 38.0 |
| 3 | 16 | 3 | 3 | 4.4 | 5.0 | 5/5 | 30.2 | 52.0 |
| 4 | 12 | 3 | 3 | 3.0 | 3.2 | 5/5 | 25.2 | 37.4 |
| 4 | 16 | 3 | 3 | 5.6 | 6.8 | 5/5 | 34.6 | 52.6 |

size $M$ increases with $k$, but the logical depth is not monotone. This reflects the fact that, on small random instances, resource counts depend not only on $M$, but also on the detailed sparse state-preparation schedules generated from the observed $k$-mer distributions.

### 4.6 Grover-iteration scaling

Table 3 isolates the cost of increasing the number of Grover iterations for the fixed setting $k = 2, L = 8, N = 4$ and $r = 3$ QAE evaluation qubits. As expected, the logical depth, $T$-count, and rotation count grow approximately linearly with the number of Grover iterations, since each Grover iteration repeats the marking and diffusion components of the coherent search circuit.

The logical depth increases from 615.0 with no Grover iterations to 5549.7 with four Grover iterations. Similarly, the rotation count increases from 663.0 to 5967.0. This approximately linear growth confirms that the Q# logical resource estimates reflect the repeated application of the Grover search iterate.

## 5 Conclusion

We introduced a quantum framework for alignment-free closest-pair search in biological sequence collections using distributional $k$-mer statistics. The central observation is that square-root amplitude encoding maps the Bhattacharyya coefficient between empirical $k$-mer distributions to the inner product between the corresponding quantum states. This makes the squared Bhattacharyya score accessible through a SWAP-test-based QAE procedure. The procedure is used in the finite-precision marking circuit of our proof-of-concept workflow and motivates the cost assigned to coherent comparisons in the separate oracle-level analysis.

Our main result is an oracle-level theorem. Under a consistent $\varepsilon$-resolved coherent-comparison-oracle assumption, Dürr–Høyer maximum finding returns, with probability at least $2/3$, a pair whose score is at least $F^\star - \varepsilon$, using $O(N)$ expected comparison-oracle applications. An optimal-score gap greater than $\varepsilon$ implies exact recovery.

**Table 2** Q# logical resource estimates obtained using `qsharp.logical_counts` for circuits with $r = 3$ QAE evaluation qubits and one Grover iteration. Each row reports the mean over three random trials. Here $M = |\mathcal{K}|$ is the observed $k$-mer basis size, $R_{\mathrm{tot}} = \sum_i R_i$ is the total sparse state-preparation schedule size, and the remaining columns are logical resource quantities reported by Q#.

| $k$ | $L$ | $N$ | $M$ | $R_{\mathrm{tot}}$ | Qubits | Depth | $T$-count | Rotations |
|---|---|---|---|---|---|---|---|---|
| 2 | 5 | 3 | 8.7 | 12.7 | 29.7 | 668.0 | 978.0 | 735.0 |
| 2 | 8 | 3 | 12.7 | 22.3 | 31.0 | 1264.3 | 1518.0 | 1395.0 |
| 2 | 8 | 4 | 14.7 | 31.7 | 31.0 | 1848.7 | 1926.0 | 1989.0 |
| 2 | 8 | 5 | 14.3 | 38.0 | 35.0 | 2323.0 | 1950.0 | 2415.0 |
| 2 | 8 | 6 | 14.7 | 47.3 | 35.0 | 3175.3 | 2322.0 | 3315.0 |
| 2 | 8 | 7 | 15.7 | 54.0 | 35.0 | 3483.7 | 2682.0 | 3627.0 |
| 2 | 8 | 8 | 15.3 | 61.7 | 35.0 | 4201.3 | 2766.0 | 4389.0 |
| 2 | 12 | 3 | 14.7 | 31.3 | 31.0 | 2322.3 | 1698.0 | 2475.0 |
| 2 | 16 | 3 | 14.7 | 33.7 | 31.0 | 3398.3 | 1758.0 | 3555.0 |
| 3 | 12 | 3 | 23.3 | 39.3 | 35.0 | 1931.7 | 2778.0 | 2055.0 |
| 3 | 16 | 3 | 31.0 | 54.3 | 36.3 | 3292.7 | 3078.0 | 3495.0 |
| 4 | 12 | 3 | 26.3 | 38.0 | 35.0 | 1609.7 | 2778.0 | 1695.0 |
| 4 | 16 | 3 | 38.7 | 59.0 | 39.0 | 2864.7 | 3498.0 | 3135.0 |

**Table 3** Effect of the number of Grover iterations on Q# logical resource estimates for the setting $k = 2$, $L = 8$, $N = 4$, and $r = 3$ QAE evaluation qubits. The logical depth, $T$-count, and rotation count grow approximately linearly with the number of Grover iterations.

| Grover iters | Logical qubits | Logical depth | $T$-count | Rotations |
|---|---|---|---|---|
| 0 | 25.0 | 615.0 | 642.0 | 663.0 |
| 1 | 31.0 | 1848.7 | 1926.0 | 1989.0 |
| 2 | 31.0 | 3082.3 | 3210.0 | 3315.0 |
| 4 | 31.0 | 5549.7 | 5778.0 | 5967.0 |

The resulting gate bounds are conditional on both the efficient consistent-order implementation assumed in Assumption 3 and the chosen data-access model. Combining this comparison-order assumption with idealized qRAM-style indexed loading gives a sequential gate complexity of $\widetilde{O}\left(\frac{NL}{\varepsilon}\right)$. Combining the same comparison-order assumption with explicit multiplexed indexed loading instead gives $\widetilde{O}\left(\frac{N^2 L}{\varepsilon}\right)$.

The current Q# implementation validates the finite-precision loading, QAE, marking, and Grover-search components, but does not establish the execution-wide consistency assumption used by the main theorem. It should therefore be viewed as a circuit-level proof of concept that makes the coherent loading cost explicit.

Several directions remain for future work. One direction is to give a complete circuit-level realization of the consistent comparison-oracle assumption and to analyze comparison consistency near the additive-resolution boundary. A second direction is to develop more efficient coherent indexed-loading mechanisms, including qRAM-style or hybrid data structures that reduce the cost of loading sparse $k$-mer distribution states while accounting for routing, coherence, and fault-tolerant overheads [11, 3]. Another direction is to move beyond the black-box maximum-finding formulation by exploiting biological or statistical structure in $k$-mer distributions, sequence databases, or induced feature spaces. Finally, compact classical

sketches such as MinHash reduce memory and runtime by approximately preserving similarity, and future quantum or hybrid methods could study whether sketch-derived features can be represented as quantum states and compared using overlap-estimation primitives.

Together, these directions suggest a path toward quantum and hybrid algorithms for alignment-free omics search that combine coherent similarity estimation, maximum finding, and practical compressed sequence representations.

## References


1 Scott Aaronson, Nai-Hui Chia, Han-Hsuan Lin, Chunhao Wang, and Ruizhe Zhang. On the quantum complexity of closest pair and related problems. In *35th Computational Complexity Conference (CCC 2020)*, volume 169 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:43. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.CCC.2020.16.

2 Farid Ablayev, Kamil Khadiev, Alexander Vasiliev, and Mansur Ziiatdinov. Theory and applications of quantum hashing. *Quantum Reports*, 7(2):24, 2025. doi:10.3390/quantum7020024.

3 Srinivasan Arunachalam, Vlad Gheorghiu, Tomas Jochym-O'Connor, Michele Mosca, and Priyaa Varshinee Srinivasan. On the robustness of bucket brigade quantum RAM. *New Journal of Physics*, 17:123010, 2015. doi:10.1088/1367-2630/17/12/123010.

4 Afrad Basheer, A. Afham, and Sandeep K. Goyal. Quantum $k$-nearest neighbors algorithm. *arXiv preprint*, 2020. arXiv:2003.09187.

5 Anil Bhattacharyya. On a measure of divergence between two statistical populations defined by their probability distribution. *Bulletin of the Calcutta Mathematical Society*, 35:99–110, 1943.

6 Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. Tight bounds on quantum searching. *Fortschritte der Physik*, 46(4–5):493–505, 1998. arXiv:quant-ph/9605034.

7 Gilles Brassard, Peter Hoyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. *arXiv preprint quant-ph/0005055*, 2000.

8 Harry Buhrman, Richard Cleve, John Watrous, and Ronald De Wolf. Quantum fingerprinting. *Physical Review Letters*, 87(16):167902, 2001.

9 Christoph Dürr and Peter Høyer. A quantum algorithm for finding the minimum. *arXiv preprint quant-ph/9607014*, 1996.

10 Robert C. Edgar. Search and clustering orders of magnitude faster than BLAST. *Bioinformatics*, 26(19):2460–2461, 2010. doi:10.1093/BIOINFORMATICS/BTQ461.

11 Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Architectures for a quantum random access memory. *Physical Review A*, 78(5):052310, 2008. doi:10.1103/PhysRevA.78.052310.

12 Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Quantum random access memory. *Physical Review Letters*, 100(16):160501, 2008. doi:10.1103/PhysRevLett.100.160501.

13 Niels Gleinig and Torsten Hoefler. An efficient algorithm for sparse quantum state preparation. In *Proceedings of the 58th ACM/IEEE Design Automation Conference*, pages 433–438, 2021. doi:10.1109/DAC18074.2021.9586240.

14 Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing (STOC)*, pages 212–219, 1996. doi:10.1145/237814.237866.

15 Lov K. Grover and Terry Rudolph. Creating superpositions that correspond to efficiently integrable probability distributions. *arXiv preprint quant-ph/0208112*, 2002.

16 Peter Høyer, Michele Mosca, and Ronald de Wolf. Quantum search on bounded-error inputs. In *Automata, Languages and Programming*, volume 2719 of *Lecture Notes in Computer Science*, pages 291–299. Springer, 2003. doi:10.1007/3-540-45061-0_25.

17 Weizhong Li, Lukasz Jaroszewski, and Adam Godzik. Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences. *Bioinformatics*, 22(13):1658–1659, 2006. doi:10.1093/BIOINFORMATICS/BTL158.

18 Guillaume Marçais, Dan DeBlasio, and Carl Kingsford. Sketching methods with small window guarantee using minimum decycling sets. *Journal of Computational Biology*, 31(7):597–615, 2024. doi:10.1089/CMB.2024.0544.

19 Brian D. Ondov, Gregory J. Starrett, Nicholas H. Sappington, et al. Mash Screen: high-throughput sequence containment estimation for genome discovery. *Genome Biology*, 20:232, 2019.

20 Brian D. Ondov, Todd J. Treangen, Péter Melsted, et al. Mash: fast genome and metagenome distance estimation using MinHash. *Genome Biology*, 17:132, 2016.

21 Yihui Quek, Clément L. Canonne, and Patrick Rebentrost. Robust quantum minimum finding with an application to hypothesis selection. *arXiv preprint*, 2020. arXiv:2003.11777.

22 Debora Ramacciotti, Andreea-Iulia Lefterovici, and Antonio F. Rotundo. A simple quantum algorithm to efficiently prepare sparse states. *Physical Review A*, 110:032609, 2024.

23 Torbjørn Rognes, Tomáš Flouri, Ben Nichols, Christopher Quince, and Frédéric Mahé. VSEARCH: a versatile open source tool for metagenomics. *PeerJ*, 4:e2584, 2016.

24 Karthik C. S. and Pasin Manurangsi. On closest pair in euclidean metric: Monochromatic is as hard as bichromatic. In *10th Innovations in Theoretical Computer Science Conference (ITCS 2019)*, volume 124 of *Leibniz International Proceedings in Informatics*, pages 17:1–17:16, 2019. doi:10.4230/LIPIcs.ITCS.2019.17.

25 Michael Ian Shamos and Dan Hoey. Closest-point problems. In *16th Annual Symposium on Foundations of Computer Science (sfcs 1975)*, pages 151–162. IEEE, 1975. doi:10.1109/SFCS.1975.8.

26 Jim Shaw and Yun William Yu. Fast and robust metagenomic sequence comparison through sparse chaining with skani. *Nature Methods*, 20:1661–1665, 2023.

27 Krysta M. Svore, Alan Geller, Matthias Troyer, John Azariah, Christopher Granade, Bettina Heim, Vadym Kliuchnikov, Mariia Mykhailova, Andres Paz, and Martin Roetteler. Q#: enabling scalable quantum computing and development with a high-level domain-specific language. *arXiv preprint*, 2018. arXiv:1803.00652.

28 Nathan Wiebe, Ashish Kapoor, and Krysta M. Svore. Quantum algorithms for nearest-neighbor methods for supervised and unsupervised learning. *Quantum Information and Computation*, 15(3–4):316–356, 2015. doi:10.26421/QIC15.3-4-7.

29 Shengli Zhang and Tianming Wang. A novel alignment-free method for phylogenetic analysis of protein sequences. In *Proceedings of the 10th WSEAS International Conference on Applied Computer Science*, pages 67–71, 2010.

30 Hongyu Zheng, Guillaume Marçais, and Carl Kingsford. Creating and using minimizer sketches in computational genomics. *Journal of Computational Biology*, 30(12):1251–1276, 2023. doi:10.1089/CMB.2023.0094.

# Exact and Efficient Inference of Tumor Phylogenies via Novel Pruning Techniques

**Juan Luque**
Department of Computer Science, University of Maryland, College Park, MD, USA

**Jacob Gilbert**
Cancer Data Science Laboratory, National Cancer Institute, Bethesda, MD, USA

**Arjun Subramanian**
The Pingry School, Basking Ridge, NJ, USA

**Aravind Srinivasan***
Department of Computer Science, University of Maryland, College Park, MD, USA

**Salem Malikic***
Cancer Data Science Laboratory, National Cancer Institute, Bethesda, MD, USA

**S. Cenk Sahinalp***
Cancer Data Science Laboratory, National Cancer Institute, Bethesda, MD 20894, USA

## Abstract

Reconstructing the evolutionary history of tumors using single-cell sequencing (SCS) data presents significant computational challenges. Existing approaches are either computationally intractable for emerging large-scale datasets or rely on heuristics that lack optimality guarantees. In this work, we propose a novel, time-efficient algorithm that constructs the phylogenetic tree of tumor evolution with a provable guarantee of optimality.

Our main result is a branch-and-bound algorithm that reconstructs the most likely tumor evolutionary history up to two orders of magnitude faster than the previous best algorithm. To achieve this, we use efficient and well-known 2-approximation algorithms for the Vertex Cover problem to prune the branch-and-bound tree effectively. Unlike previous works' polynomial-time branch-and-bound bounding strategies, our bounding algorithm provides strong worst-case theoretical guarantees, leading to faster reconstruction of the tumor evolution.

**2012 ACM Subject Classification** Theory of computation → Design and analysis of algorithms

**Keywords and phrases** Branch and Bound, Vertex Cover, Linear Programming, Tumor Evolution, Single-Cell Sequencing

**Digital Object Identifier** 10.4230/LIPIcs.WABI.2026.10

**Supplementary Material** *Software*: `https://github.com/jdluque/faster-tphyl-reconstruction`

**Funding** S.C.S., S.M. and J.G. were supported by the Intramural Research Program of the National Cancer Institute (NCI), National Institutes of Health.

## 1 Introduction

Cancer is an evolutionary disease, driven by the progressive accumulation of somatic mutations, which give rise to genetically distinct subclones within a tumor. A history of tumor progression can be represented as a phylogenetic tree of tumor evolution, which captures the lineage relationships among tumor cells and provides insights into early driver mutations, patterns of metastatic seeding, mutually exclusive and co-occurring mutations, and clonal expansions.

* Joint last authors

© Juan Luque, Jacob Gilbert, Arjun Subramanian, Aravind Srinivasan, Salem Malikic, and S. Cenk Sahinalp;
licensed under Creative Commons License CC-BY 4.0

The inference of trees of tumor evolution has been one of the very important problems in cancer genomics research. Early methods predominantly relied on bulk sequencing data, which provides trees of limited resolution and has well known limitations [11, 14]. However, advances in single-cell sequencing (SCS) technologies have enabled much more detailed studies of tumor evolution. Consequently, there has been rapid progress in the development of computational methods for inferring trees of tumor evolution from this type of data.

Existing methods for inferring tumor evolutionary trees from SCS data can broadly be classified into two main categories. The first category consists of heuristic-based approaches, including techniques such as Monte Carlo Markov Chain (MCMC) sampling [10, 17, 21, 26, 23], nearest-neighbor interchange [25], and greedy search [12]. While these methods typically return some solution tree, they do not offer any guarantees of optimality for the reported solutions. Their notable limitations include terminating at locally optimal but globally suboptimal solutions, and, in some cases, sensitivity to minor variations in the input (e.g., permutations in the order of cells in the input can lead to different outputs, despite the inputs being effectively equivalent).

To address some of the limitations discussed above, a second category of methods has been developed. These methods are typically based on Integer Linear Programming (ILP) [16, 7, 19, 6, 5], and rely on established ILP solvers such as Gurobi or CPLEX. Unlike heuristic-based approaches, ILP-based methods can offer guarantees of optimality when they successfully converge to optimal solutions. However, they are often prohibitively slow on larger datasets containing hundreds of mutations or cells, which significantly limits their practical applicability on emerging single-cell datasets. To mitigate this scalability issue, some methods [7] apply pre-clustering of cells as a preprocessing step. While this can reduce computational complexity, it often comes at the cost of reduced accuracy and/or lower resolution in the inferred evolutionary trees [12].

One potential direction for improving the efficiency of ILP-based methods involves designing specialized algorithms that exploit structural properties of the tumor phylogeny inference problem. The goal of such algorithms is to accelerate the solution of the underlying ILP without sacrificing accuracy. A notable example of this approach is the recently developed PhISCS-BnB, which leverages branch and bound techniques to efficiently search the solution space [18]. PhISCS-BnB has been shown to match the accuracy of other ILP-based methods on smaller datasets, while outperforming them on larger instances. In particular, for some large-scale inputs, it was the only method capable of producing an optimal solution (along with a certificate of optimality) within a reasonable time.

While PhISCS-BnB marked a significant advancement over previous state-of-the-art methods in terms of performance, it continued to face a critical limitation: in the worst case, it may explore an exponential number of search tree nodes, solving an NP-complete problem at each node. This limitation constrains its scalability, especially in the context of increasingly large and complex datasets. As a result, the development of more effective pruning strategies – particularly those with *polynomial-time* complexity – remained a key open challenge.

Building upon the principled framework and formal optimality guarantees introduced in PhISCS-BnB, our work advances the field by enabling even faster phylogeny reconstruction. Specifically, we replace the computationally intensive SAT solvers previously used for branch-and-bound pruning with efficient, polynomial-time Linear Programming (LP) solvers. Furthermore, we provide optimality guarantees for our newly introduced pruning algorithms, laying the groundwork for continued improvements in both the scalability and robustness of phylogeny inference methods.

Notably, the use of PDLP [1], Google's freely-available recent Linear Programming (LP) solver, enabled our algorithms to achieve the fastest phylogeny reconstruction times in our experimental evaluations. In this context, our work not only initiates research into provably more powerful pruning algorithms for phylogeny reconstruction but is also expected to benefit from ongoing advancements in LP solvers.

## 2 Background and problem definition

We assume that a single-cell sequencing experiment has been conducted, in which single cells $C = \{C_1, C_2, \ldots, C_n\}$ were sampled from a tumor and sequenced. We also assume that, after the mutation calling step, we are given a set of mutations $M = \{M_1, M_2, \ldots, M_m\}$ that are reported to be present in at least one of these cells. Therefore, our input can be summarized as an *observed genotype matrix*, denoted in this work as $I$, consisting of $n$ rows (corresponding to cells) and $m$ columns (corresponding to mutations). In this matrix, $I_{ip} = 1$ indicates that the mutation calling step reported mutation $M_p$ to be present in cell $C_i$, while $I_{ip} = 0$ indicates its absence. Given $I$, our goal is to reconstruct the most parsimonious evolutionary history of the tumor.

Before providing a rigorous mathematical definition of the computational problem we aim to solve, we first present the model of evolution used in this work and define how we represent the evolutionary history of a tumor. To simplify our presentation, we focus on a hypothetical case in which the ground truth is available. In the ground truth, cells from $C$ and mutations from $M$ define the *ground truth genotype matrix* $T$, which, analogous to $I$, is a binary matrix of size $n \times m$, with 0 and 1 respectively indicating the absence and presence of a mutation in a cell. An example of such a matrix is shown in Figure 1a.

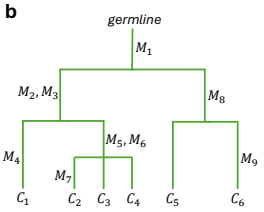
In this work, we assume the commonly used Infinite Sites Assumption (ISA), which states that each mutation is acquired at most once during the course of tumor evolution and, once acquired by a cell, is never lost in any of its descendants. This assumption implies that tumor evolution follows the Perfect Phylogeny Model, in which mutations are inherited along the edges of a rooted tree. More precisely, under the ISA, the ground truth matrix $T$ defines a unique *tree of tumor evolution* (referred to as the tree in the remainder of this paper, unless otherwise specified), which is a rooted tree consisting of $n$ leaves, each labeled with a distinct cell from $C$. The root of the tree represents the germline (i.e., normal cells), which we assume contains no somatic mutations. The edges are labeled with mutations such that each edge that is not incident to a leaf or the root contains at least one mutational label, whereas the other edges may or may not contain such labels.[1] A mutation assigned to the edge connecting a node $v$ to its parent is assumed to be present in the cell labeling $v$ (if $v$ is a leaf), or in all cells labeling the descendant leaves of $v$ (if $v$ is an internal node). This implies that the set of mutations harbored by cell $C_i$ can be obtained by taking the union of all mutations appearing on the unique path connecting the leaf labeled by $C_i$ to the root of the tree. Note that, for each cell, the presence/absence pattern of mutations in that cell is consistent between the ground truth matrix $T$ and what is implied by the tree corresponding to $T$. In general, when this property holds for a binary matrix and a tree of tumor evolution, we say that the tree and the matrix are *consistent*. The tree consistent with the ground truth matrix shown in Figure 1a is presented in Figure 1b.

[1] Note that the ground truth evolutionary history of the sequenced cells is a binary tree; however, in our case, we may not have sufficient signal to reconstruct a fully resolved binary tree from $T$. In other words, the trees used in this work may contain polytomies (i.e., internal nodes with more than two children).

**a**

| | $M_1$ | $M_2$ | $M_3$ | $M_4$ | $M_5$ | $M_6$ | $M_7$ | $M_8$ | $M_9$ |
|---|---|---|---|---|---|---|---|---|---|
| $C_1$ | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| $C_2$ | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 |
| $C_3$ | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| $C_4$ | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| $C_5$ | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| $C_6$ | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

**b**




**c**

| | $M_1$ | $M_2$ | $M_3$ | $M_4$ | $M_5$ | $M_6$ | $M_7$ | $M_8$ | $M_9$ |
|---|---|---|---|---|---|---|---|---|---|
| $C_1$ | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| $C_2$ | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 |
| $C_3$ | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| $C_4$ | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| $C_5$ | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $C_6$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

**Figure 1** **a**. Ground truth genotype matrix. **b**. Tree of tumor evolution consistent with the ground truth matrix shown in (a). **c**. Observed genotype matrix obtained from real single-cell sequencing data. This matrix typically differs from the ground truth (red entries) and often contains conflicts, which imply that no tree is consistent with the matrix. An example of such a conflict is highlighted in yellow.

While it is straightforward to reconstruct the tree consistent with the ground truth matrix (and this can be done in polynomial time using the algorithm introduced in [9]), the ground truth matrix is unknown in practice. As discussed above, what we are given instead as input is an observed genotype matrix $I$ obtained from a single-cell sequencing experiment. An example of such a matrix is shown in Figure 1c. Due to sequencing noise and mutation calling errors, the observed genotype matrix usually differs from the ground truth. With existing single-cell sequencing technologies, the most prevalent type of noise in $I$ is false negative mutation calls, where a mutation $M_p$ that is present in the ground truth in cell $C_i$ (i.e., $T_{ip} = 1$) is reported as absent in $I$ (i.e., $I_{ip} = 0$). Although the observed matrix $I$ may also contain some false positives, their occurrence is typically rare – with reported rates ranging between 0.0001 and 0.001 [8]. Previous work has shown that such low false positive rates have minimal impact on the accuracy of the reconstructed trees when using methods that explicitly model only false negatives, such as the approach introduced in [18].[2] Given this, and in line with [18], we assume that the input data in this work contains only false negatives. Nevertheless, we also assess the performance of our method under realistic levels of false positives and show that it remains largely robust to this type of noise.

Due to the presence of erroneous mutation calls, there typically does not exist a tree that is consistent with the observed matrix $I$. More precisely, it is well known that any binary matrix $H$ containing a triplet of rows $(i, j, h)$ and a pair of columns $(p, q)$ such that

$$\begin{bmatrix} H_{ip} & H_{iq} \\ H_{jp} & H_{jq} \\ H_{hp} & H_{hq} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$$

is not consistent with any tree under the Perfect Phylogeny Model [15]. We say that such a pair of mutations and triplet of cells forms a *conflict*, and we refer to binary matrices that contain no conflicts as *perfect phylogeny matrices*. It is well known that for every perfect phylogeny matrix there exists a tree consistent with it [9].

Given the presence of conflicts in the observed matrix $I$, existing methods for reconstructing tumor evolutionary trees aim to correct $I$ by flipping some of its entries to obtain a perfect phylogeny matrix, from which a tree can be reconstructed. These methods approach the

[2] Alternatively we can handle a small ($\leq c$ for a constant $c$) number of false positives by solving our problem for every subset of $\leq c$ columns removed from $I$ [16] and returning the solution with the best objective. While a serial implementation of this approach could increase the running time by a factor exponential in $c$, a trivial parallelization would result in a speedup linear with the number of processors.

problem either explicitly, by directly operating in the space of binary matrices, or implicitly, by searching in the space of trees and scoring them using the perfect phylogeny genotype matrices they imply (see [15] for more details).

This leads us to the formal definition of the *Perfect Phylogeny Reconstruction Problem* that we aim to solve in this work.

▶ **Definition 1** (Perfect Phylogeny Reconstruction Problem). *Given a binary genotype matrix $I$, determine the minimum set of zero entries in $I$ whose conversion to ones yields a perfect phylogeny matrix.*

For brevity, we use PP to denote *perfect phylogeny matrix* and "most likely PP" to denote the *most likely perfect phylogeny matrix*; i.e., the perfect phylogeny matrix obtained by flipping a minimum number of zero entries in the observed matrix.

The most likely perfect phylogeny problem has been studied before under the name of the i-flip variant of the minimum-flip problem [3]. Here the authors prove NP-completeness and give an $O(2^k mn)$ fixed-parameter tractable (FPT) algorithm for the decision problem "does there exist a perfect phylogeny requiring at most $k$ flips?" However, translating this FPT algorithm to the most likely perfect phylogeny problem would lead to exploring the full, unpruned search tree considered in this work. This would lead to an algorithm orders of magnitude slower and is not feasible at the size of input considered in this work. There are approximate and FPT algorithms for the minimum-flip problem allowing both types of flips [3, 2, 22]. However, these methods allow both one-to-zero and zero-to-one flips with equal weight, which is not well suited for single-cell sequencing data where false positive errors are orders of magnitude rarer than false negative errors. Direct ILP formulations for variants of this minimum flip problem have been studied in [16, 4], but these approaches' scalability is limited by the need to solve large ILPs.

## 3 Preliminaries

Throughout the remainder of the paper, let $H$ denote an arbitrary binary matrix, which, depending on the context, may represent either the original input matrix $I$ or an intermediate matrix generated during the branch-and-bound procedure. As an intermediary step towards the full perfect phylogeny reconstruction problem, we attempt to get a handle on $\mathrm{OPT}_{\mathrm{init}}(H)$, the minimum number of flips required to remedy all *initial* conflicts in $H$, disregarding any potential newly introduced conflicts (i.e., all initial conflicts are resolved if for each row triplet with configuration (0,1), (1,0), (1,1), at least one of the (0,1) or (1,0) is flipped to a (1,1)). Note that the details on computing $\mathrm{OPT}_{\mathrm{init}}(H)$ will be provided later (e.g., in Section 4.2). Analogously, let $\mathrm{OPT}(H)$ be the minimum number of flips necessary to make $H$ a PP. Observe that, since a PP obtained by flipping some entries of $H$ must resolve all initial conflicts in $H$, as well as any conflicts introduced in the process of resolving those initial conflicts, we always have $\mathrm{OPT}_{\mathrm{init}}(H) \leq \mathrm{OPT}(H)$. Next, a binary matrix $D$ is said to be a *descendant* of a binary matrix $H$ if $D$ can be obtained by flipping some zeros in $H$. Additionally, define $F_{0\to 1}(H, D)$ as the number of flips required to turn $H$ into $D$ and $\mathrm{PP}(H) = \{D : D \text{ is a descendant of } H \text{ and } D \text{ is a PP}\}$. Let $R_{01}(p, q) = \{i : H_{ip} = 0 \wedge H_{iq} = 1\}$; i.e. the set of $(0, 1)$ rows in columns $p$ and $q$. Similarly, define $R_{10}(p, q) = \{i : H_{ip} = 1 \wedge H_{iq} = 0\}$ and $R_{11}(p, q) = \{i : H_{ip} = 1 \wedge H_{iq} = 1\}$. Note that, strictly speaking, the sets $R_{01}(p, q)$, $R_{10}(p, q)$, and $R_{11}(p, q)$ should also include the matrix $H$ in the notation, but we omit this dependence throughout the paper for notational simplicity.

## 3.1 Vertex Cover approximation algorithms

*Vertex Cover* (VC) is a fundamental problem in combinatorial optimization. Given an undirected graph $(V, E)$, where $V$ is the set of vertices and $E$ is the set of edges, the goal of VC is to find a subset $C \subseteq V$ of minimum cardinality such that for every edge $(u, v) \in E$, at least one of the endpoints $u$ or $v$ belongs to $C$. VC is NP-complete and cannot be solved in polynomial-time unless P = NP. Nevertheless, VC admits efficient 2-approximation algorithms; or 2-approximations, for short. That is, VC admits algorithms that, in polynomial-time, return a vertex cover $C$ with cardinality provably at most twice the cardinality of a minimum vertex cover $C^*$; i.e., $|C| \leq 2|C^*|$. VC has a very rich history and admits two remarkably simple 2-approximations that we make use of in this work. Interested readers can find detailed treatment of these 2-approximations in any combinatorial optimization reference; e.g., [20].

The first 2-approximation finds a maximal matching $M$ in time $O(|E|)$ by simply iterating over each $(u, v) \in E$ and adding $(u, v)$ to $M$ if both endpoints are unmatched. In the end, it returns $C = \bigcup_{(u,v)\in M} \{u, v\}$. The second VC 2-approximation formulates VC as an LP.

$$\begin{aligned} \min \quad & \sum_{v \in V} x_v & & & \text{(LP-VC)} \\ \text{subj. to} \quad & x_u + x_v \geq 1, & & \forall (u, v) \in E & (1) \\ & x_v \geq 0, & & \forall v \in V & (2) \end{aligned}$$

Next, solutions $x^* \in [0, 1]^{|V|}$ of the above LP are rounded to $X^* \in \{0, 1\}^{|V|}$, where, for each $v \in V$, $X^*_v = 1$ if $x^*_v \geq 1/2$ and $X^*_v = 0$ otherwise. Finally, the algorithm returns the vertex cover $C = \{v \in V : X^*_v = 1\}$. (LP-VC) is well known to be *half-integral*, meaning that there always exists an optimal solution $x^*$ satisfying $x^*_v \in \left\{0, \frac{1}{2}, 1\right\}$ for all $v \in V$.

## 3.2 Branch-and-bound and PhISCS-BnB

In order to prove optimality of the PP returned by PhISCS-BnB when given an input matrix $I$, the authors lean on the branch-and-bound paradigm. Branch-and-bound builds a search tree where each node corresponds to some binary matrix $H$, so for brevity we refer to search tree nodes and matrices interchangeably. The tree's root node is the input matrix $I$, its leaves are precisely PPs, and, for any $D$ in the subtree rooted at $H$, $D$ is a descendant of $H$. An internal node $H'$ is not a PP and has parent node $H$. Here, $H'$ is obtained by flipping zeros to fix all conflicts in $H$ between some pair of columns $p$ and $q$. In an optimal solution, conflicts between columns $p$ and $q$ must be fixed by either flipping all the zeros in column $p$ and rows $R_{01}(p, q)$ or all the zeros in column $q$ and rows $R_{10}(p, q)$.

By exploring this tree, branch-and-bound either explores or proves suboptimality for all PPs in PP$(I)$. Given a matrix $H$, a (possibly suboptimal) PP $B$, and a lower bound $L(H)$ on the required number of flips to make $H$ a PP, then all matrices $D$ that are descendants of $H$ can be ruled out (i.e., "pruned") if $F_{0\to1}(I, H) + L(H) \geq F_{0\to1}(I, B)$. In this event, no PP $D$, which is a descendant of $H$, can beat $B$ and therefore may be pruned out of the search tree. When an explored leaf beats $B$, it becomes the new best node and the value of $B$ is updated; [18] also describes simple heuristics for finding an initial guess $B$. In this way, PhISCS-BnB takes a binary matrix $H$ with conflicts, gradually introduces flips to explore a tree of all possibly optimal descendants while pruning nodes that can be ruled out, and, finally, terminates when it has found and certified a most likely PP.

**Branch-and-bound algorithm.** Our branch-and-bound algorithm, provided in detail in Algorithm 1, builds on top of PhISCS-BnB. The key differences are the use of our new bounding algorithms (Section 4.2) and an additional step in which we check whether the "free" byproduct PPs produced by our algorithms improve upon the current incumbent PP, updating it when they do (Section 4.6).

### 3.2.1 Lower bounding mechanisms

PhISCS-BnB explores two formulations for its lower bounding mechanism by solving related instances of i) *minimum weighted SAT* (MWS) and ii) *maximum weight matching* (MWM). Despite MWS itself being NP-hard, and thus requiring super-polynomial time in the worst case, in practice, PhISCS-BnB with MWS terminates orders of magnitude faster than PhISCS-BnB with MWM.

**MWM lower bounding mechanism.** Given a matrix $H$, the MWM lower bounding mechanism constructs a graph $(V, E)$ where vertices correspond to columns of $H$ and edges correspond to column pairs with conflicts. Formally, MWM sets up a graph $(V, E)$ with $V = \{p : 1 \leq p \leq m\}$ and $E = \{(p, q) : |R_{01}(p, q)| \geq 1 \wedge |R_{10}(p, q)| \geq 1 \wedge |R_{11}(p, q)| \geq 1\}$. Observe that, for any pair of columns $p$ and $q$ of matrix $H$, a conflict is implied by the presence of all three value pairs: $(0, 1)$, $(1, 0)$, and $(1, 1)$, when considering the values of these columns across all rows of $H$. Since we are only allowed to make flips from 0 to 1, resolving all conflicts between columns $p$ and $q$ requires flipping all 0's in column $p$ to 1 in each row of $R_{01}(p, q)$ *or* flipping all 0's in column $q$ and each row of $R_{10}(p, q)$. In other words, we need at least the smaller of the two numbers of flips in order to resolve all conflicts between columns $p$ and $q$. Hence, for each $e = (p, q) \in E$ the edge weight $w_e$ is set to $\min\{|R_{01}(p, q)|, |R_{10}(p, q)|\}$ when $|R_{11}(p, q)| > 0$ and $w_e = 0$ otherwise. The matching property ensures that flips that may fix multiple conflicts are not counted multiple times. Let MWM($H$) be the weight of such a MWM in the corresponding $(V, E)$. Indeed, the weight of any matching $M$ on $(V, E)$ is a lower bound on $\text{OPT}_{\text{init}}(H)$ since $w_e$ for $e = (p, q)$ is a lower bound on the number of flips from columns $p$ or $q$ that a PP must make to resolve all conflicts between $p$ and $q$. Therefore, MWM($H$) is the best matching-based lower bound.

**MWS lower bounding mechanism.** Given a matrix $H$, the *minimum weighted SAT* (MWS) lower bounding mechanism writes a weighted SAT instance wherein the weight of a minimum satisfying assignment lower bounds OPT. PhISCS-BnB considers other MWS formulations. For brevity's sake, we refer interested readers to [18]. Like VC, MWS is NP-complete. However, unlike our algorithms, which solve VC approximately and in polynomial time, PhISCS-BnB solves MWS instances exactly in worst case super-polynomial time.

## 3.3 Outline of our contributions

We make key observations relating the perfect phylogeny reconstruction problem to vertex cover and use these insights to derive branch-and-bound lower bounding mechanisms inspired by well-known 2-approximation algorithms of vertex cover. In Section 4, we show that our new polynomial-time lower bounding mechanisms achieve provably stronger guarantees than the polynomial-time lower bounding mechanisms of PhISCS-BnB.

In Section 5, we benchmark our algorithm on synthetic datasets. Our experiments demonstrate that the proposed polynomial-time lower bounding mechanisms significantly accelerate branch-and-bound search, yielding speedups of up to 1000x over PhISCS-BnB.

While our best performing lower bounding algorithms can only account for initial conflicts, in Section 4.4, we introduce an additional lower bounding mechanism that gives a provably stronger lower bound that accounts for conflicts beyond only those that are initially present.

**Algorithm 1** VC-PhISCS-BnB: Branch and bound method.

**Input:** Original input matrix $I \in \{0,1\}^{n \times m}$
$B \leftarrow$ A simple PP (see Section 4.7) ; `// Best incumbent node`
$Q \leftarrow$ An empty priority queue;
Push $I$ into $Q$;
**while** $Q$ *is not empty* **do**
  $C \leftarrow$ pop next node from $Q$ with minimum priority score;
  $p, q \leftarrow$ columns of $C$ in conflict;
  `// Branch into two new nodes,` $S^{(1)}$ `and` $S^{(2)}$`, by fixing the` $pq$ `conflict by flipping either` $(0,1)$`s or` $(1,0)$`s`
  $S^{(1)} \leftarrow C$, $S^{(2)} \leftarrow C$;
  **for** $r \in R_{01}(p,q)$ **do**
    $S^{(1)}_{r,p} \leftarrow 1$;
  **end**
  **for** $r \in R_{10}(p,q)$ **do**
    $S^{(2)}_{r,q} \leftarrow 1$;
  **end**
  **for** $i \in \{1,2\}$ **do**
    **if** $S^{(i)}$ *is a PP* **then**
      **if** $F_{0\to1}\left(I, S^{(i)}\right) < F_{0\to1}\left(I, B\right)$ **then**
        `//` $S^{(i)}$ `beats incumbent best node` $B$
        $B \leftarrow S^{(i)}$;
      **end**
    **else**
      `// Explore nodes that cannot be pruned away`
      $L \leftarrow$ lower bound on the number of flips to make $S^{(i)}$ a PP; e.g. from Section 4.2;
      **if** $L + F_{0\to1}\left(I, S^{(i)}\right) < F_{0\to1}\left(I, B\right)$ **then**
        Push $S^{(i)}$ in $Q$ with priority score set to $L + F_{0\to1}\left(I, S^{(i)}\right)$;
      **end**
      `// Use "free" upper bound, if available, to update best incumbent node` $B$`; see Section 4.6`
      **if** *Lower bounding algorithm gives a "free" incumbent best node* **then**
        $B' \leftarrow$ candidate best node byproduct;
        **if** $F_{0\to1}\left(I, B'\right) < F_{0\to1}\left(I, B\right)$ **then**
          $B \leftarrow B'$;
        **end**
      **end**
    **end**
  **end**
**end**
$Y \leftarrow B$;
**return** $Y$;

# 4 Lower bounding algorithms

## 4.1 Provably better polynomial-time lower bounds

To further motivate the search for better polynomial-time lower bound mechanisms, we take a closer look at MWM. Let MWM($H$) be the maximum weight matching computed on the graph corresponding to matrix $H$. Let $S_{n,m}$ be the $n \times m$ binary matrix consisting of $n-2$ copies of the row $(0,1,1,\ldots,1)$, one $(1,0,0,\ldots,0)$ row, and one $(1,1,\ldots,1)$ row; i.e.,

$$S_{n,m} = \begin{bmatrix} 0 & 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 1 & 1 & \cdots & 1 \\ 1 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 1 & \cdots & 1 \end{bmatrix}.$$

Observe that all conflicts in $S_{n,m}$ occur between column 1 and another column $p > 1$. Therefore, the corresponding MWM graph of $S_{n,m}$ is a star graph with center vertex corresponding to the first column. Similarly, because each conflict can be resolved via a single flip in the second to last row, all edge weights are 1. Hence, by these two facts, $\text{MWM}(S_{n,m}) = 1$. However, the optimal solution uses $\min\{n-2, m-1\}$ flips, indicating flipping all zeros in either the first column or the second to last row, respectively.

▶ Remark 2. For all $k \geq 3$, $\text{MWM}(S_{k,k+2}) = 1$ despite $\text{OPT}_{\text{init}}(S_{k,k+2}) = k-2$.

Even as $\text{OPT}_{\text{init}}$ increases, MWM lower bounds may provide no information at all. A lower bound of 1 is merely an unhelpful "the current matrix is not a PP" message.

## 4.2 Vertex Cover formulation and lower bounding algorithms

Our algorithms take a binary matrix $H$ and use the VC approximations described in Section 3. Let $Z(H)$ be a set of coordinates for pairs of zeros appearing in conflicts together; i.e.

$$Z(H) = \{((i,p),(j,q)) : (1 \leq p < q \leq m) \wedge (i \in R_{01}(p,q)) \wedge (j \in R_{10}(p,q)) \wedge (|R_{11}(p,q)| > 0)\}.$$

Define the graph $(V_H, E_H)$ with $V_H = \{(i,p) \mid H_{ip} = 0\}$ and $E_H = Z(H)$; for comparison, $H$ has $nm$ entries, $|V| \leq nm$, and $|E| = O(n^2m^2)$. Algorithm 5 (provided in the Appendix) directly uses the maximal matching-based approximation on $(V_H, E_H)$. Similarly, Algorithm 3, uses the LP-based approximation where LP-VC is written taking $(V,E) = (V_H, E_H)$.

▶ **Lemma 3.** *$C \subseteq V_H$ is a VC of $(V_H, E_H)$ if, and only if, flipping $H_{ip}$ for all $(i,p) \in C$ fixes all initial conflicts of $H$.*

**Proof.** Observe that: a conflict between $(i,p)$ and $(j,q)$ in $Z(H)$ is fixed $\iff$ $H_{i,p}$ or $H_{j,q}$ is flipped $\iff$ $(i,p) \in C$ or $(j,q) \in C$ $\iff$ $((i,p),(j,q)) \in E$ is covered. As $Z(H)$ and $E_H$ are in a one-to-one correspondence, the claim follows. ◀

▶ **Corollary 4.** *Let $C^*$ be a minimum VC of $(V_H, E_H)$. Then $|C^*| = \text{OPT}_{\text{init}}(H)$.*

Behind effective pruning algorithms, lie effective "lower bounding" algorithms. That is, given a matrix $H$, we need algorithms that produce strong lower bounds on the number of flips necessary to make $H$ a PP. As seen in Remark 2, the MWM bound is not useful in the worst case. Unlike MWM, our lower bounding algorithms remain provably useful.

▶ **Theorem 5.** *Let $L_{VC}$ and $L_{LP\text{-}VC}$ be the lower bound values returned by Algorithms 3 and 5, respectively, on a binary matrix $H$. Then $\text{OPT}_{\text{init}}(H)/2 \leq L_{VC}, L_{LP\text{-}VC} \leq \text{OPT}_{\text{init}}(H)$.*

**Proof.** $L_{\text{VC}} = \lceil |C|/2 \rceil$ where $|C|$ is a VC of $(V_H, E_H)$, $|C| \leq 2|C^*|$ for a minimum VC $C^*$. Applying Corollary 4 and the definition of 2-approximation,

$$\text{OPT}_{\text{init}}(H) = |C^*| \leq |C| \leq 2|C^*| = 2\,\text{OPT}_{\text{init}}(H).$$

Thus, dividing by two, $\text{OPT}_{\text{init}}(H)/2 \leq |C|/2 \leq \text{OPT}_{\text{init}}$. Since $\text{OPT}_{\text{init}}(H)$ is an integer, we can take a ceiling and complete the proof for the claim about $L_{\text{VC}}$.

For the next lower bound, $L_{\text{LP-VC}} = \left\lceil \sum_{(i,p)\in V_H} x^*_{ip} \right\rceil$ with $x^*$ and $X^*$ being the fractional and rounded LP solutions, respectively.

$$\sum_{i,p} x^*_{ip} \leq |C^*| = \text{OPT}_{\text{init}}(H) \leq \sum_{i,p} X^*_{ip} \leq 2 \sum_{i,p} x^*_{ip}.$$

The first inequality follows from the well known fact that $L_{\text{LP-VC}} \leq |C^*|$, the equality follows from Corollary 4, the second inequality follows from the fact that $\{(i,p) : X^*_{ip} = 1\} \subseteq V_H$ defines a VC, and, finally, the last inequality follows from the fact that $X^*_{ip} \leq 2x^*_{ip}$ for all $(i,p) \in V_H$. Like before, $\text{OPT}_{\text{init}}(H)$ is an integer, therefore we can take a ceiling and both bounds on $L_{\text{LP-VC}}$ follow. ◀

Between Algorithms 3 and 5, the fastest (and simplest) approach is arguably the maximal matching-based Algorithm 5. Since both algorithms rely on VC 2-approximations, why not stick to the simpler one? Computation time is only one aspect of an effective lower bounding algorithm; quality of the lower bound matters too. For the simple VC approach, the lower bound, $|C|/2$, and the VC size, $|C| \leq 2\,\text{OPT}_{\text{init}}$, are, pessimistically, always a factor of 2 apart. Meanwhile, for (LP-VC), instead of the analogous lower bound $\frac{1}{2}\sum_{i,p} X^*_{ip} = |C|/2$, we can use a greater $\sum_{i,p} x^*_{ip}$.

▶ Remark 6. $\sum_{i,p} x^*_{ip} - \frac{1}{2}\sum_{i,p} X^*_{ip} = \frac{N}{2}$ where $N = |\{(i,p) : x^*_{ip} = 1\}|$.

**Proof.** By the half-integrality of (LP-VC) we can split the summation as

$$\sum_{i,p;x^*_{ip}\in\left\{0,\frac{1}{2}\right\}} \left(x^*_{ip} - \frac{1}{2}X^*_{ip}\right) + \sum_{i,p;x^*_{ip}=1} \left(x^*_{ip} - \frac{1}{2}X^*_{ip}\right) = \sum_{i,p;x^*_{ip}=1} \left(x^*_{ip} - \frac{1}{2}X^*_{ip}\right) = \frac{N}{2}. \quad ◀$$

In practice, a substantial $N/2$ enables Algorithm 3 to prune the search tree more aggressively and may even offset the additional computational cost of writing and solving an LP. This tradeoff between lower bound quality and computation speed is explored once more in the last of our lower bounding algorithms, Algorithm 4.

**Algorithm 2** Check if a set of flips makes the matrix a PP.

**Input:** $H$ is the matrix at the current branch-and-bound node, $F$ is a set of flips resolving all initial conflicts on $H$, $H \oplus F$ returns $H$ after applying the flips of $F$.

**function** *CandidateBestNode*$(H, F)$
  $B \leftarrow$ Node requiring $\infty$ flips to become a PP;
  **if** $H \oplus F$ *is a PP* **then**
    $B \leftarrow H \oplus F$;
  **end**
  **return** $B$
**end**

**Algorithm 3** Lower bound using (LP-VC).

**Input:** $H$ is the matrix at the current branch-and-bound node and `solver` is an LP solver.
**function** *LPLowerBound*($H$, *solver*)
  $(V_H, E_H) \leftarrow$ VC instance from $H$ as described in Section 4.2;
  $\Pi \leftarrow$ (LP-VC) instance written using $(V, E) = (V_H, E_H)$;
  $x^* \leftarrow$ optimal solution to $\Pi$, obtained using `solver`;
  **for** $H_{ip} = 0$ **do**
    $X^*_{ip} \leftarrow 1$ if $x^*_{ip} \geq \frac{1}{2}$ else 0;
  **end**
  $C \leftarrow \{(i,p) : X^*_{ip} = 1\}$;
  **return** $\left\lceil \sum_{i,p} x^*_{ip} \right\rceil$, *CandidateBestNode*($H, C$);
**end**

## 4.3 Considering only the initial conflicts in $I$

We now focus on Algorithm 4, stated fully in the Appendix, which is also an LP-based approach but deals with (LP-VC) defined by $(V, E) = (V_H, E_H \cap E_I)$ where $I$ is the initial input matrix. Like before, $E_I = Z(I)$, so $E_I \cap E_H$ is a set of edges corresponding to conflicts that are present in *both* $I$ and $H$. The compromise between computational speed and lower bound quality is as follows. We write the full (LP-VC) once with the graph $(V_I, E_I)$. Then, at each node of the branch-and-bound tree, we only *partially* rewrite (LP-VC). That is, at each node, instead of rewriting the costly $O(m^2n^2)$ LP constraints, we just update the lower bounds of $O(mn)$ variables to ensure that the entries that have been flipped between $I$ and $H$ remain flipped. This results in a significant speedup, as $mn \ll (mn)^2$; owing to the high quality of LP solvers available, writing the LP itself can be much more time consuming than solving it. On the other hand, this approach weakens our lower bound as it will not account for conflicts introduced in the flips from $I$ to $H$. An analogous guarantee to Theorem 5 holds for Algorithm 4.

▶ Remark 7. The lower bound of Algorithm 4 is at least half the number of flips necessary to fix conflicts present in both $I$ and $H$.

## 4.4 Going beyond initial conflicts

The lower bounds on the number of flips we prove in Theorem 5 hits a ceiling at initial conflicts. That is, for any binary matrix $H$ and lower bound $L(H)$ on the number of flips necessary to make $H$ a PP from Theorem 5, $L(H) \leq \mathrm{OPT}_{\mathrm{init}}(H)$. In this section we introduce a new LP formulation giving stronger lower bounds that can exceed $\mathrm{OPT}_{\mathrm{init}}(H)$. Let $\mathcal{I} := \{(0,1),(1,0),(1,1)\}$ and define our LP as follows.

$$\min \quad \sum_{i,p:H_{ip}=0} x_{ip} \qquad \text{(LP-Extended)}$$

$$\text{subj. to} \quad -x_{ip} + x_{iq} \leq B(p,q,0,1), \qquad \forall 1 \leq i \leq n,\ 1 \leq p < q \leq m \qquad (3)$$

$$x_{ip} - x_{iq} \leq B(p,q,1,0), \qquad \forall 1 \leq i \leq n,\ 1 \leq p < q \leq m \qquad (4)$$

$$x_{ip} + x_{iq} \leq 1 + B(p,q,1,1), \qquad \forall 1 \leq i \leq n,\ 1 \leq p < q \leq m \qquad (5)$$

$$\sum_{(a,b)\in\mathcal{I}} B(p,q,a,b) \leq 2, \qquad \forall 1 \leq p < q \leq m \qquad (6)$$

$$0 \leq B(p,q,a,b) \leq 1, \qquad \forall 1 \leq p < q \leq m,\ (a,b) \in \mathcal{I} \qquad (7)$$

$$0 \leq x_{ip} \leq 1, \qquad \forall H_{ip} = 0 \qquad (8)$$

This LP is reminiscent of the integer program in [16], which is used for a closely related problem. Like in (LP-VC), the variable $x_{ip}$ corresponds to the matrix entry $H_{ip}$, with $x_{ip} = 1$ denoting that $H_{ip}$ should be flipped and $x_{ip} = 0$ saying to leave $H_{ip}$ unflipped. The auxiliary variables $B(p, q, a, b)$ ensure, for each $(a, b) \in \mathcal{I}$, that if columns $p$ and $q$ have a row with values $(a, b)$, then $B(p, q, a, b) = 1$. Lastly, Constraints (6) ensure an integer solution corresponds to a matrix that is a PP.

Given a matrix $H$, let $L_1 = \sum_{i,p} x^*_{ip}$ and $L_2 = \sum_{i,p} y^*_{ip}$ be LP objectives at optimal solutions of (LP-VC) and (LP-Extended), respectively.

▶ **Lemma 8.** $L_1(H) \leq L_2(H)$.

▶ **Lemma 9.** $L_2(H) \leq \mathrm{OPT}(H)$.

**Proof.** Let $H^*$ be a most likely PP. It suffices to show that $x^*$ with $x^*_{ip} = H^*_{ip} \in \{0, 1\}$ is in the feasible region of (LP-Extended). As $H^*$ is a PP, for any pair of columns $p$ and $q$, the columns $p$ and $q$ do not contain (at least) one of the three rows $(0, 1)$, $(1, 0)$, or $(1, 1)$; otherwise $H^*$ would not be a PP. This means for every pair $p$ and $q$, there exists an assignment of $B(p, q, a, b)$, for $(a, b) \in \mathcal{I}$ such that Constraints (3), (4), (5), and (6) are all satisfied. Therefore, $x^*$ is in the feasible region of (LP-Extended). ◀

▶ **Lemma 10.** *There exist binary matrices $U$ where $L_2(U) > \mathrm{OPT}_{\mathrm{init}}(U) \geq L_1(U)$.*

Note that proofs of Lemma 8 and Lemma 10 can be found in the Appendix. Finally, the following theorem follows from Lemmas 8–10.

▶ **Theorem 11.** $L_1(H) \leq L_2(H) \leq \mathrm{OPT}(H)$. *Moreover, there exist binary matrices $U$ for which $L_2(U) > \mathrm{OPT}_{\mathrm{init}}(U)$.*

On the other hand, this LP does not always perform better. Let $I_{n,n}$ be the $n \times n$ identity matrix and let $\mathbf{1}$ be the matrix of size $n \times n$ with all of its entries equal to 1. The matrix $\mathbf{1} - I_{n,n}$ will have the associated VC instance be a complete graph on $n$ vertices. Therefore, $\mathrm{OPT} = n - 1$ but both LP objectives will be $\mathrm{OPT}/2$.

## 4.5 Trading off between LP formulation and speed

We identify that rewriting an LP to account for the conflicts present in every node of the BnB search tree is a bottleneck. To this end, in Section 4.3 we consider only the initial conflicts of $I$ in its computed lower bound. At each node of the search tree we only need to adjust the bounds of $O(mn)$ variables, instead of rewriting the much larger LP. However, $H$ may have introduced new conflicts. To consider these conflicts without needing to also rewrite the LP from scratch, we use (LP-Extended) in Section 4.4.

Alongside Algorithm 4, this lets us consider all initial conflicts at every matrix $H$, while avoiding the need to fully rewrite the LP at every node of the search tree. At the center of this tradeoff lies a question whose answer depends on the given input matrix $I$: "what fraction of the conflicts that must be resolved are present initially?" If few or no conflicts are to be re-introduced, then the larger LP is not worth the additional computation. However, if conflicts are introduced frequently enough, then the additional computation of (LP-Extended) can be worth it.

### 4.6 Upper bounding algorithms at low computational cost

There is one more ingredient to effectively pruning nodes: providing a good "best node" $B$ guess (i.e. a possibly suboptimal PP), which promises the optimal solution can only take *at most* $F_{0\to 1}(I, B)$. In this way each PP $B$ gives an *upper bound* on the number of required flips. In addition to lower bounds, our algorithms also use the byproduct VCs to check whether they lead to a new best node. In particular, if the flips suggested by the VC introduce no new conflicts, then the resulting matrix constitutes a PP, and we check whether this new PP improves upon the best one seen so far. This logic for updating the best-so-far node lies in Algorithm 2; checking whether a matrix is a PP is done in $O(nm)$ time using the algorithm of [9].

### 4.7 Hybrid algorithms

In many synthetic matrices, the MWS lower and upper bounding heuristic of PhISCS-BnB is able to terminate at the root node of the search tree. In other words, the lower bound produced at the root node matches the best node.

To take advantage of this, we build a hybrid branch-and-bound algorithm by borrowing this strategy only for the first best node guess. Then we use our own algorithms from Section 4 for the remainder of the branch-and-bound algorithm. Analogous to Algorithm 4, this is yet another algorithmic compromise between quality and speed of lower bounds that performs promisingly.

Polynomial time can be guaranteed for the initial guess by running both the MWS formulation and our polynomial-time algorithms in parallel and taking the better guess of the two after some (polynomial) time. As all of the lower bounding algorithms suggest a set of flips that solve initial conflicts, we can simply run the lower bounding algorithm, perform the suggested flips, and repeat until we achieve a PP. This is guaranteed to converge in polynomial-time because if the current matrix has any conflicts, then some nonzero number of flips will be suggested and the matrix of all ones is a PP.

## 5 Results

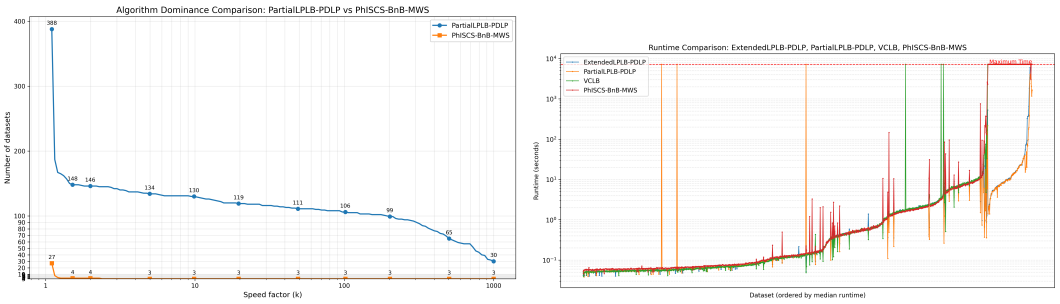
### 5.1 Experimental setup

All experiments presented in this section were run using Python 3.10 on an 8-core machine with an Intel Xeon Silver 4216 CPU and 64GB of RAM. The branch-and-bound logic is implemented using the library `pybnb`. Our code repository forks the code of [18] and is available at `https://github.com/jdluque/faster-tphyl-reconstruction`. To implement the various lower bounding mechanisms we use the Python libraries `Cython`, `numpy`, `NetworkX`, and `pysat`. LPs are solved using Google's PDLP [1] through their open sourced `or-tools` package as well as Gurobi. We distinguish which lower bounding algorithm was used to branch-and-bound in the following way. LPLowerBound(Solver) uses Algorithm 3 and PartialLPLowerBound(PDLP) uses Algorithm 4, each using one of PDLP or Gurobi as the LP solver. ExtendedLPLB(PDLP) denotes Algorithm 4 using (LP-Extended) and PDLP as the underlying LP solver. VCLB uses Algorithm 5 and is run with $r = 5$ repetitions. All of our presented algorithms are the hybrid variants as discussed in Section 4.7. PhISCS-BnB using MWS and MWM as the underlying lower bounding algorithm are denoted by PhISCS-BnB-MWS and PhISCS-BnB-MWM, respectively.

To assess the performance of our algorithm, we simulated phylogenetic trees representing tumor evolution with a specified number of cells $n \in \{20, 30, 50, 100\}$, each labeling a distinct leaf of the simulated tree (see Figure 1b). We then randomly assigned a given number of

mutations $m \in \{20, 30, 50, 100, 250, 500, 1000\}$ to the edges of the tree. This tree gives us a unique ground truth genotype matrix with $400 \leq mn \leq 100,000$ entries. To generate the noisy (observed) genotype matrix $I$ used as input to our method, we introduced noise into the ground truth matrix as follows: for each entry equal to 1, we flipped it to 0 with probability $p \in \{0.05, 0.10, 0.15, 0.20\}$ (i.e., $p$ denotes the false negative rate). For each combination of parameters $(m, n, p)$ we generated 10 distinct instances.

In addition to the simulated datasets described above, we also evaluated our methods on the same mouse melanoma dataset used in PhISCS-BnB, derived from the study of [24] and available through Python's `scphylo-tools` library.




**(a)** Runtime comparison of PartialLPLB(PDLP) and PhISCS-BnB-MWS.

**(b)** Runtime comparison extended to include VCLB and ExtendedLPLB-PDLP.

**Figure 2** **(a)** The runtime of each algorithm for the specified dataset on a log scale with different matrices at each value of the horizontal axis. Matrices are sorted by the median time to find the most likely PP between the competing algorithms. **(b)** Additionally considers algorithms VCLB and ExtendedLPLB-PDLP. The lines that go all the way up to the timeout line indicate a dataset that the algorithm failed to complete.

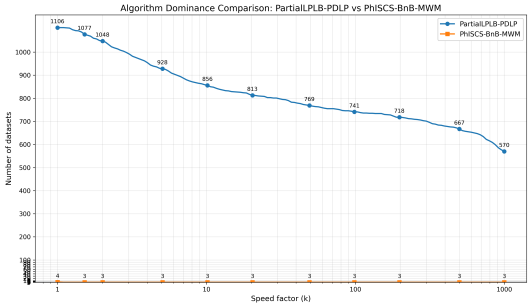



**Figure 3** Comparison of speedup factors of PartialLPLB(PDLP) and PhISCS-BnB-MWM shows that our new method is a vast improvement over the previous best polynomial-time lower bound algorithm.

## 5.2 Experimental results

Each algorithm is allotted 48 hours total to work through the 1,120 generated matrices. Further, each algorithm is allowed a maximum of 2 hours per matrix before timing out. For the sake of clearer plots, we rank instances by the median time required for the various algorithms to compute a most likely PP.

A natural comparison benchmark is then to ask how many PP each algorithm finds within time constraints. This result is displayed in Table 1 with our LP-based methods being the clear winners, finding 98.8% of PPs in the largest matrices with $mn = 100,000$ entries.

These are followed by PhISCS-BnB-MWS and VCLB, which find 89.5% and 89.0% of PPs on matrices with as many as $mn = 50,000$ entries. Finally, all algorithms greatly outperform PhISCS-BnB-MWM which only finds PPs for 39.7% of matrices including matrices with up to $mn = 3,000$ entries.

We plot the speedup factor $k$, as well as the number of datasets for which this speedup factor is achieved (i.e., the number of matrices in which one algorithm finishes $k$ times faster than the other).

Bringing up the rear, PhISCS-BnB-MWM finds PPs for 445 matrices with up to $nm = 3000$ entries. Next, VCLB and PhISCS-BnB-MWS, find PPs for 997 and 1002 matrices with up to $mn = 50000$ entries, respectively. Leading the pack, PartialLPLB(PDLP), ExtendedLPLB(Gurobi), ExtendedLPLB(PDLP) all find PPs for 1107 matrices with $mn = 100,000$ entries.

**Table 1** Performance of various methods in finding PPs within time constraints.

| Method | PPs found | Max Entries ($mn$) |
|---|---|---|
| **ExtendedLPLB(PDLP)** | **1107** (=98.8%) | **100,000** |
| **PartialLPLB(PDLP)** | **1107** (=98.8%) | **100,000** |
| **PartialLPLB(Gurobi)** | **1107** (=98.8%) | **100,000** |
| PhISCS-BnB-MWS | 1002 (=89.5%) | 50,000 |
| VCLB | 997 (=89.0%) | 50,000 |
| PhISCS-BnB-MWM | 445 (=39.7%) | 3,000 |

Our greatest improvements over PhISCS-BnB-MWS were generally achieved by PartialLPLB(PDLP) and ExtendedLPLB(PDLP), as shown in Figures 2 and 4 (note that Figure 4 is provided in the Appendix). Intuitively, it is not always optimal to disregard conflicts that are introduced later on, as PartialLPLB(PDLP) does. Still, this proves to be a valuable variant, as PartialLPLB(PDLP) outperforms ExtendedLPLB(PDLP) in many instances, as shown in Figure 4. On the other hand, ExtendedLPLB(PDLP) performs better on some of the harder instances, so there is no conclusive winner. Across the board, no algorithm outperformed every other algorithm on every dataset, as shown in Figure 2b. Relative to PhISCS-BnB-MWS, ExtendedLPLB(PDLP) achieves speedup factors as large as 1000x on 27 instances. The improvements are much steeper compared with the previous polynomial-time lower bounds of PhISCS-BnB-MWM. In Figure 3, PartialLPLB(PDLP) is orders of magnitude faster than PhISCS-BnB-MWM across nearly all instances.

As suggested by the number of solved instances in Table 1, VCLB also consistently outperforms PhISCS-BnB-MWM. Compared to PhISCS-BnB-MWS, VCLB has an edge on the easier instances, but does not keep up; as the difficulty increases, it times out more often making its overall performance slightly worse than PhISCS-BnB-MWS. Although VCLB is simpler and the bounding algorithm itself is pretty fast, this speedup does not offset the time cost of exploring more nodes of the search tree as a result of the lower bound's deterioration. This mirrors the discussion at the end of Section 4.2. Interestingly, PartialLPLB with PDLP and Gurobi perform comparably, with PDLP completing a few more instances faster, as shown in Figure 5. Lastly, we benchmarked our methods on the mouse melanoma dataset introduced above. On this dataset, our algorithms and PhISCS-BnB-MWS both reconstruct the most likely PP in 80 minutes.

### 5.3 Tree reconstruction accuracy in the presence of false positives

We also evaluated the performance of our method in the presence of false positive mutation calls in the input data. To this end, we conducted additional experiments in which, alongside false negatives, we also simulated false positives. Given that false positives typically occur at lower rates in practice [8], we simulated them at rates of 0.01, 0.001, and 0.0001.

As shown in Figure 6, PartialLPLB(PDLP) remains largely robust to the presence of false positives, with ancestor–descendant accuracy decreasing only slightly relative to the corresponding experiments without false positives. More specifically, the average ancestor–descendant accuracy at false positive rates of 0.01, 0.001, and 0.0001 were 0.984, 0.975, and 0.9240, respectively, corresponding to decreases between 0.01 and 0.06 compared to the baseline without false positives. By contrast, the reduction in different-lineage accuracy was even smaller, with decreases of less than 0.0001 relative to the baseline without false positives across all evaluated false positive rates. Since the presence of false positives resulted in only negligible changes in different-lineage accuracy, we do not show separate plots for these results.

Overall, these results demonstrate that our method is only marginally affected by the presence of false positives and remains highly accurate under realistic levels of such noise.

## 6 Discussion and future work

Although the latter lower bound derived in (LP-Extended) is stronger than that of (LP-VC), experiments showed that the lower bound from (LP-VC) solved the perfect phylogeny reconstruction problem faster. In fact, (LP-Extended) is asymptotically smaller using the same number of variables but $O(nm^2)$ constraints compared to the $O(n^2m^2)$ constraints of (LP-VC). This finding is yet another example of an important recurring theme in this work: theory versus practice; or better yet, theory-aided practice.

Nevertheless, our theory-informed algorithms saw speedups upwards of two orders of magnitude in experiments. Stronger theoretical results, such as the bounds of (LP-Extended) in Section 4.4, will likely continue to be an important part of future improvements. This bound may also see more success with even larger datasets or in settings where the SCS data noise is more adversarial; for example, noise with correlations instead of uniformly random.

Our work is a practical advancement for the most likely perfect phylogeny reconstruction problem and sets up groundwork for future improvements by providing the first polynomial-time pruning algorithms with theoretical guarantees. Additionally, we leave room to improve our algorithm via the, still unexploited, polynomial-time pruning algorithms that account for more conflicts than just those initially present in a matrix, as well as with practical improvements in LP solvers such as [13].

## References


1 David Applegate, Oliver Hinder, Haihao Lu, and Miles Lubin. Faster first-order primal-dual methods for linear programming using restarts and sharpness. *Mathematical Programming*, 201(1-2):133–184, September 2023. doi:10.1007/s10107-022-01901-9.

2 Sebastian Böcker, Quang Bao Anh Bui, and Anke Truss. Improved Fixed-Parameter Algorithms for Minimum-Flip Consensus Trees. *ACM Trans. Algorithms*, 8(1):7:1–7:17, January 2012. doi:10.1145/2071379.2071386.

3 Duhong Chen, O. Eulenstein, D. Fernandez-Baca, and M. Sanderson. Minimum-flip supertrees: complexity and algorithms. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(2):165–173, April 2006. doi:10.1109/TCBB.2006.26.

4 Markus Chimani, Sven Rahmann, and Sebastian Böcker. Exact ILP solutions for phylogenetic minimum flip problems. In *Proceedings of the First ACM International Conference on Bioinformatics and Computational Biology*, pages 147–153, Niagara Falls New York, August 2010. ACM. doi:10.1145/1854776.1854800.

5 Simone Ciccolella, Mauricio Soto Gomez, Murray D Patterson, Gianluca Della Vedova, Iman Hajirasouliha, and Paola Bonizzoni. gpps: an ilp-based approach for inferring cancer progression with mutation losses from single cell data. *BMC bioinformatics*, 21:1–16, 2020.

6 Mohammadamin Edrisi, Hamim Zafar, and Luay Nakhleh. A combinatorial approach for single-cell variant detection via phylogenetic inference. In *19th International Workshop on Algorithms in Bioinformatics (WABI 2019)*, pages 22:1–22:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.WABI.2019.22.

7 Mohammed El-Kebir. Sphyr: tumor phylogeny estimation from single-cell sequencing data under loss and error. *Bioinformatics*, 34(17):i671–i679, 2018. doi:10.1093/BIOINFORMATICS/BTY589.

8 Yusi Fu, Chunmei Li, Sijia Lu, Wenxiong Zhou, Fuchou Tang, X Sunney Xie, and Yanyi Huang. Uniform and accurate single-cell sequencing based on emulsion whole-genome amplification. *Proceedings of the National Academy of Sciences*, 112(38):11923–11928, 2015.

9 Dan Gusfield. Efficient algorithms for inferring evolutionary trees. *Networks*, 21(1):19–28, 1991. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/net.3230210104. doi:10.1002/net.3230210104.

10 Katharina Jahn, Jack Kuipers, and Niko Beerenwinkel. Tree inference for single-cell data. *Genome Biology*, 17(1), May 2016. doi:10.1186/s13059-016-0936-x.

11 Jack Kuipers, Katharina Jahn, and Niko Beerenwinkel. Advances in understanding tumour evolution through single-cell sequencing. *Biochimica et Biophysica Acta (BBA)-Reviews on Cancer*, 1867(2):127–138, 2017.

12 Can Kızılkale, Farid Rashidi Mehrabadi, Erfan Sadeqi Azer, Eva Pérez-Guijarro, Kerrie L. Marie, Maxwell P. Lee, Chi-Ping Day, Glenn Merlino, Funda Ergün, Aydın Buluç, S. Cenk Sahinalp, and Salem Malikić. Fast intratumor heterogeneity inference from single-cell sequencing data. *Nature Computational Science*, 2(9):577–583, September 2022. Publisher: Nature Publishing Group. doi:10.1038/s43588-022-00298-x.

13 Haihao Lu and Jinwen Yang. cuPDLP.jl: A GPU Implementation of Restarted Primal-Dual Hybrid Gradient for Linear Programming in Julia, June 2024. arXiv:2311.12180 [math]. doi:10.48550/arXiv.2311.12180.

14 Salem Malikic, Katharina Jahn, Jack Kuipers, S. Cenk Sahinalp, and Niko Beerenwinkel. Integrative inference of subclonal tumour evolution from single-cell and bulk sequencing data. *Nature Communications*, 10(1):2750, June 2019. Publisher: Nature Publishing Group. doi:10.1038/s41467-019-10737-5.

15 Salem Malikić, Farid Rashidi Mehrabadi, Erfan Sadeqi Azer, Mohammad Haghir Ebrahimabadi, and Suleyman Cenk Sahinalp. Studying the history of tumor evolution from single-cell sequencing data by exploring the space of binary matrices. *Journal of Computational Biology*, 28(9):857–879, 2021. doi:10.1089/CMB.2020.0595.

16 Salem Malikic, Farid Rashidi Mehrabadi, Simone Ciccolella, Md Khaledur Rahman, Camir Ricketts, Ehsan Haghshenas, Daniel Seidman, Faraz Hach, Iman Hajirasouliha, and S. Cenk Sahinalp. PhISCS: a combinatorial approach for subperfect tumor phylogeny reconstruction via integrative use of single-cell and bulk sequencing data. *Genome Research*, 29(11):1860–1877, November 2019. doi:10.1101/gr.234435.118.

17 Edith M Ross and Florian Markowetz. Onconem: inferring tumor evolution from single-cell sequencing data. *Genome biology*, 17:1–14, 2016.

18 Erfan Sadeqi Azer, Farid Rashidi Mehrabadi, Salem Malikić, Xuan Cindy Li, Osnat Bartok, Kevin Litchfield, Ronen Levy, Yardena Samuels, Alejandro A. Schäffer, E. Michael Gertz, Chi-Ping Day, Eva Pérez-Guijarro, Kerrie Marie, Maxwell P. Lee, Glenn Merlino, Funda Ergun, and S. Cenk Sahinalp. PhISCS-BnB: a fast branch and bound algorithm for the perfect tumor phylogeny reconstruction problem. *Bioinformatics (Oxford, England)*, 36(Suppl_1):i169–i176, July 2020. doi:10.1093/bioinformatics/btaa464.

19 Gryte Satas, Simone Zaccaria, Geoffrey Mon, and Benjamin J Raphael. Scarlet: single-cell tumor phylogeny inference with copy-number constrained mutation losses. *Cell systems*, 10(4):323–332, 2020.

20 A. Schrijver. *Combinatorial Optimization - Polyhedra and Efficiency*. Springer, 2003.

21 Jochen Singer, Jack Kuipers, Katharina Jahn, and Niko Beerenwinkel. Single-cell mutation identification via phylogenetic inference. *Nature communications*, 9(1):5144, 2018.

22 Dekel Tsur. Faster parameterized algorithms for Bicluster Editing and Flip Consensus Tree. *Theoretical Computer Science*, 953:113796, April 2023. doi:10.1016/j.tcs.2023.113796.

23 Leah L Weber, Chuanyi Zhang, Idoia Ochoa, and Mohammed El-Kebir. Phertilizer: Growing a clonal tree from ultra-low coverage single-cell dna sequencing of tumors. *PLoS computational biology*, 19(10):e1011544, 2023. doi:10.1371/JOURNAL.PCBI.1011544.

24 Yochai Wolf, Osnat Bartok, Sushant Patkar, Gitit Bar Eli, Sapir Cohen, Kevin Litchfield, Ronen Levy, Alejandro Jiménez-Sánchez, Sophie Trabish, Joo Sang Lee, Hiren Karathia, Eilon Barnea, Chi-Ping Day, Einat Cinnamon, Ilan Stein, Adam Solomon, Lital Bitton, Eva Pérez-Guijarro, Tania Dubovik, Shai S. Shen-Orr, Martin L. Miller, Glenn Merlino, Yishai Levin, Eli Pikarsky, Lea Eisenbach, Arie Admon, Charles Swanton, Eytan Ruppin, and Yardena Samuels. UVB-Induced Tumor Heterogeneity Diminishes Immune Response in Melanoma. *Cell*, 179(1):219–235.e21, September 2019. doi:10.1016/j.cell.2019.08.032.

25 Yufeng Wu. Accurate and efficient cell lineage tree inference from noisy single cell data: the maximum likelihood perfect phylogeny approach. *Bioinformatics*, 36(3):742–750, 2020. doi:10.1093/BIOINFORMATICS/BTZ676.

26 Hamim Zafar, Anthony Tzen, Nicholas Navin, Ken Chen, and Luay Nakhleh. Sifit: inferring tumor trees from single-cell sequencing data under finite-sites models. *Genome biology*, 18:1–20, 2017.


# A Algorithms

**Algorithm 4** Lower bound while only partially rewriting (LP-VC).

**Input:** $H$ is the matrix at the current branch-and-bound node, $I$ is the initial matrix, $\Pi$ is an LP-VC written using $(V, E) = (V_I, E_I)$, and `solver` is an LP solver.

**function** PartialLPLowerBound($I, H, \Pi$, solver)

  **for** $H_{ip} = 1$ *and* $I_{ip} = 0$ **do**

    Set $x_{ip} \geq 1$ in $\Pi$;

  **end**

  $x^*_{ip} \leftarrow$ optimal solution to $\Pi$ using `solver`;

  **for** $H_{ip} = 1$ *and* $I_{ip} = 0$ **do**

    Set $x_{ip} \geq 0$ in $\Pi$ (i.e., unset the previous $x_{ip} \geq 1$);

  **end**

  **for** $H_{ip} = 0$ **do**

    $X^*_{ip} \leftarrow 1$ if $x^*_{ip} \geq \frac{1}{2}$ else 0;

  **end**

  $C \leftarrow \{(i, p) : X^*_{ip} = 1\}$;

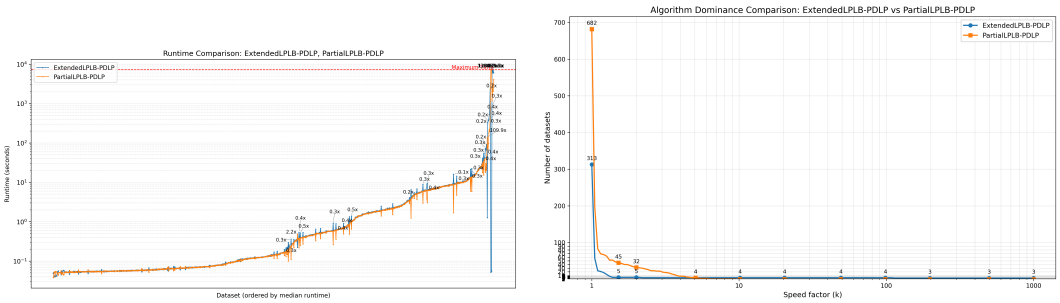
  **return** $\left\lceil \sum_{i,p} x^*_{ip} \right\rceil$, CandidateBestNode($H, C$)

**end**

**Algorithm 5** Lower bound using a VC instance.

**Input:** $H$ is the matrix at the current branch-and-bound node.
**function** *VCLowerBound*$(H)$
  $(V_H, E_H) \leftarrow$ VC instance from $H$ as described in Section 4.2;
  $C \leftarrow \emptyset$;
  **for** $((i,p),(j,q)) \in E_H$ **do**
    **if** $(i,p) \notin C$ *and* $(j,q) \notin C$ **then**
      $C \leftarrow C \cup \{(i,p),(j,q)\}$;
    **end**
  **end**
  **return** $\left\lceil \frac{|C|}{2} \right\rceil$, *CandidateBestNode*$(H, C)$
**end**

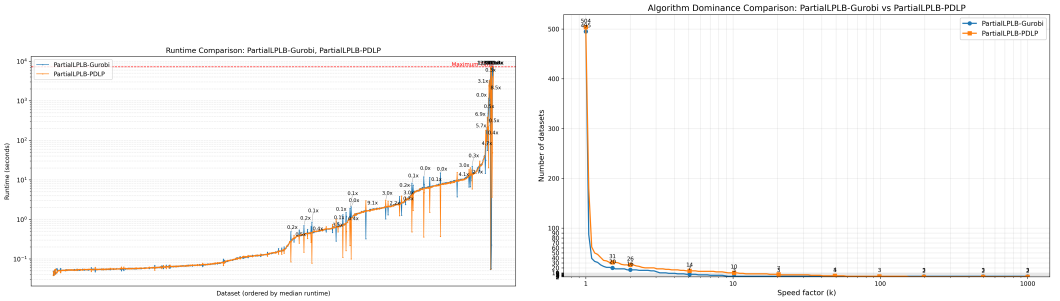
## B Additional Figures




**(a)** Speed comparison of ExtendedLPLB(PDLP) and PartialLPLB(PDLP).

**(b)** Speedup factor comparison between PartialLPLB(PDLP) and PhISCS-BnB-MWS.

**Figure 4** PartialLPLB(PDLP) slightly outperforms its Extended LP counterpart in more instances throughout, but ExtendedLPLB(PDLP) comes out on top in some of the harder instances of Figure 4a.




**(a)** Speed comparison of PartialLPLB(Gurobi) and PartialLPLB(PDLP).

**(b)** Speedup factor comparison of PartialLPLB(Gurobi) & PartialLPLB(PDLP).

**Figure 5** These plots compare the runtimes of PartialLPLB using PDLP and Gurobi as the underlying solvers. Generally, PDLP was most performant, especially as the instances take longer to solve. Figure 5b shows each speedup factor $k$, as well as a count of the number of datasets in which an algorithm finishes $k$ times faster than the other.

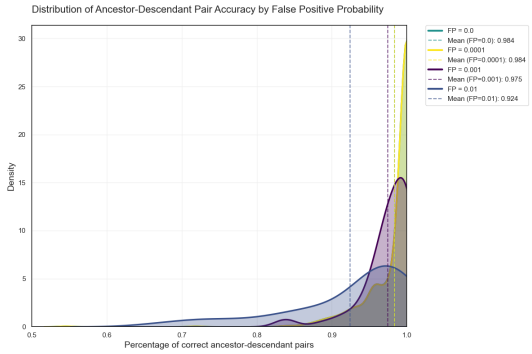



**Figure 6** Assessment of robustness in the presence of false positive noise. Ancestor–descendant accuracy is shown for varying false positive rates (for the definition of this measure, please refer to [14]).

## C Proofs

**Proof of Lemma 8.** It suffices to show that the feasible region of (LP-Extended) is a subset of (LP-VC). Let $y^*$ be a feasible solution of (LP-Extended) and consider a pair of zeros in an initial conflict, say, at coordinates $(i, p)$ and $(j, q)$. Then,

$$B(p, q, 0, 1) \geq -y^*_{ip} + y^*_{iq} = -y^*_{ip} + 1. \tag{9}$$

Similarly,

$$B(p, q, 1, 0) \geq y^*_{jp} - y^*_{jq} = 1 - y^*_{jq}. \tag{10}$$

As there is an in initial conflict between $p$ and $q$, it must be that $|R_{11}(p, q)| > 0$ so

$$B(p, q, 1, 1) = 1. \tag{11}$$

Applying (6) alongside (9), (10), and (11), gives

$$(-y^*_{ip} + 1) + (1 - y^*_{jq}) + 1 \leq 2,$$

which simplifies to $y^*_{ip} + y^*_{jq} \geq 1$. In other words, $y^*$ satisfies (1) and thus lies in the feasible region of (LP-VC). ◀

**Proof of Lemma 10.** Consider the matrix

$$U = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}.$$

We can easily verify by hand that $\mathrm{OPT}_{\mathrm{init}}(U) = 2$, corresponding to flipping the two zeros in the last two rows. With the aid of an LP solver, however, we can verify that $L_2(U) = 5$, corresponding to flipping either the first or second column, in addition to the bottom two rows. We can derive more matrices, with larger gaps between $L_2$ and $\mathrm{OPT}_{\mathrm{init}}$, by increasing the number of $(0, 1, 0)$ and $(1, 0, 0)$ rows. ◀

# FPT Learning of Sparse, Robust and Interpretable Generative Models of RNA Evolution

**Samuel Gardelle**
LIX, CNRS, École polytechnique, Institut Polytechnique de Paris, Palaiseau, France

**Laurent Bulteau**
LIGM, CNRS, Université Gustave Eiffel, Marne-la-vallée, France

**Yann Ponty**
LIX, CNRS, École polytechnique, Institut Polytechnique de Paris, Palaiseau, France

## Abstract

RNA structure modeling greatly benefits from the availability of structural homologs, associated with the presence of coevolving positions in multiple alignments. Direct Coupling Analysis (DCA) is a statistical framework for inferring significant covariations as Potts models, in a way that corrects for the transitive nature of mutual information. Various instances of DCA have been proposed over time with demonstrated ability to infer molecular contacts, yet were shown to be associated with inference algorithms that are invariably data hungry, prone to overfitting, and hindered by numerical instability. Recently, edge-activated DCA (eaDCA) has emerged as an alternative which iteratively infers couplings in a greedy manner and explicitly targets sparsity.

In this work, we revisit the inference of eaDCA models in a rigorous algorithmic setting. We circumvent the #P-hardness of computing the most promising addition/update of coupling and provide exact fixed-parameter tractable algorithms for the treewidth parameter of the coupling-induced graph. We empirically show that eaDCA models are typically associated with moderate treewidth values, and validate the practical feasibility of the method by producing, in a matter of minutes, the models associated with 41 RFAM families associated with structured non-coding families. Our results reveal good recovery rates for couplings associated with conserved base pairs from the family consensus, and enable a more systematic and robust assessment of the potential of eaDCA.

**2012 ACM Subject Classification** Applied computing → Molecular evolution; Theory of computation → Parameterized complexity and exact algorithms; Applied computing → Bioinformatics

**Keywords and phrases** RNA, Structure prediction, Evolution, DCA, Tree decomposition

**Digital Object Identifier** 10.4230/LIPIcs.WABI.2026.11

**Supplementary Material** *Software*: `https://codeberg.org/samuel-gardelle/wabi26-eaDCAstar` [6], archived at `swh:1:dir:e53ad07a87e78c5d6e48b33986c10a1c937ed706`

**Funding** *Samuel Gardelle*: École Normale Supérieure de Lyon

**Acknowledgements** The authors wish to thank Martin Weigt, Francesco Calvanese, Giovanni Peinetti, Andrea Pagnani, Ivo Hofacker and Sebstian Will for helpful discussions and advice.

## 1 Introduction

**Context.** *RiboNucleic Acids (RNAs)* are ubiquitous biomolecules that participate as first-class citizens in virtually every biological mechanism. They act both as medium to store genetic information, participate as enzymes in a variety of cellular functions (replication, regulation, sensing of metabolites...), and are widely-believed to represent a likely substrate for the origin of life [8]. Despite the prevalence of RNA transcripts in the Human genome [21], and the wide availability of RNA data produced by sequencing experiments (RNAseq) [12], our understanding of these molecules still remains partial. A workable insight into the inner workings of natural *RNA families* is essential to understand their biology, but also to inform a *rational design* of *synthetic RNAs*. The use of *therapeutic RNAs* in bio-medicine


© Samuel Gardelle, Laurent Bulteau, and Yann Ponty;
licensed under Creative Commons License CC-BY 4.0



is increasingly popular, and notoriously includes small antisense RNAs [15] and mRNA vaccines [18]. This motivates the development of design methodologies capturing statistical biases, induced by an *evolutionary pressure* to preserve RNA *function*.

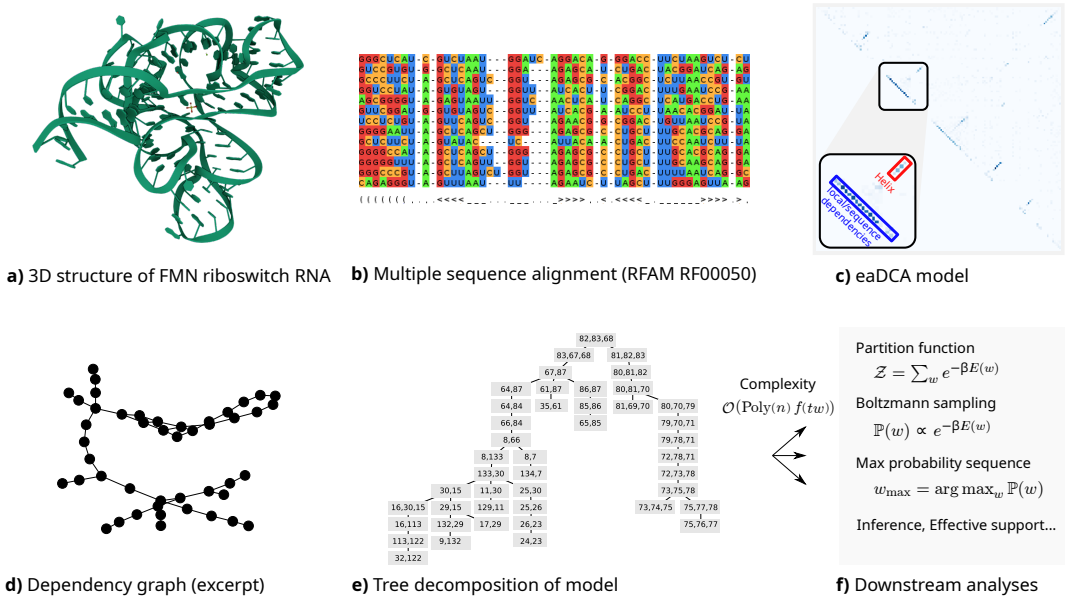
Although RNA is presented as a sequence of *nucleotides*, chemical units abstracted as A, C, G or U, the *structure* adopted by RNA is often a major determinant of its function, e.g. enabling its recognition by molecular partners. Functionally-similar RNAs, also called *homologs* [5], are often found within evolutionary-distant organisms, indicating their widespread relevance. This induces an evolutionary pressure with statistical consequences at the *nucleotide* sequence level. Notably, *base pairs*, mediated by hydrogen bonds, involving two sites will restrict their content to pairs of *complementary* nucleotides. Such a *co-evolution* can be revealed by the analysis of the mutual-information [4], informed by adequately-constructed *Multiple Sequence Alignments (MSAs)* [10].

*Direct Coupling Analysis (DCA)* [19] is a popular paradigm for an – information based – analysis of biomolecules and, more specifically, RNA families [4]. It posits that the evolutionary pressure on an RNA family can be modeled as a *Potts model*, *i.e.* a set of functions associating a weight, akin to an energy, to the content of a pair of sites (*coupling*). Such a model then induces a *Boltzmann-Gibbs distribution* on the entire space of RNA sequences (of a given length), leading to *generative models* that have been used for *molecular modeling and design* [3]. DCA models are typically inferred under the dual objective of inducing single and dual site statistics that best approximate the ones observed in MSAs, while maximizing the joint *emission probability* for RNA sequences in the family (*Maximum Likelihood Estimation; MLE*). DCA is useful to analyze co-evolution because couplings directly model co-evolving positions, e.g. strong couplings that support compatibility can typically be interpreted as evidence of proximity/connectedness on a structural level. Moreover, it has been shown experimentally [14] that it can be useful for the design of new functional RNAs.

The *interpretability* of DCA is further boosted by the typical *parsimony* of produced models, which results from DCA explicitly avoiding transitive information, and is guaranteed by design in the context of *edge-activated DCA (eaDCA)* [3]. This recently proposed version of DCA has been shown to achieve similar performances as earlier propositions, while only allowing a subset of iteratively-added edges (some possibly removed along the way) to contribute, until a predefined fraction of the single and pair statistics is induced by the model. Edges that are initially activated by eaDCA are typically part of the RNA structure, mainly located in the secondary structure, but also reveal more complex dependencies such as negative constraints preventing the adoption of alternative stable structures. However, the computational methodologies used to infer eaDCA models, sample sequences and estimate additional relevant features (e.g. *partition function*, effective support size) heavily rely on heuristics (stochastic optimization procedures; *MCMC*). Indeed, developed in the context of statistical physics/biophysics, such approaches generally do not offer guarantees of convergence, which has been identified by authors as the root cause for a loss of scalability, precision, *stability* and/or reproducibility [3].

**Contributions.** In this work, we exploit the *sparseness* of eaDCA models to design exact algorithmic methods, through the lens of *parameterized complexity*, to perform *interpretable* Machine Learning and, ultimately, decipher the key determinants of RNA molecular evolution. Our contributions include:

- Designing exact dynamic programming algorithms, having complexity parameterized by the *treewidth*, to compute the partition function and perform *statistical sampling* (*aka.* weighted random generation) in the Boltzmann-Gibbs distribution induced by a given eaDCA model;




**a)** 3D structure of FMN riboswitch RNA **b)** Multiple sequence alignment (RFAM RF00050) **c)** eaDCA model

**d)** Dependency graph (excerpt) **e)** Tree decomposition of model **f)** Downstream analyses

**Figure 1 Molecular modeling with eaDCA and tree decomposition-based algorithms.** To determine the (3D) conformation of RNA (a), co-evolving pairs of positions are extracted from a multiple sequence alignment of homologous RNA sequences (b), leading to an evolutionary coupling whose content reveals various structural features (c). Couplings can be seen as an undirected graph (d), whose tree decomposition, typically in combination with *dynamic programming*, allows to compute relevant metrics, and execute key (*NP-hard*) analyses, in time only exponential in the treewidth (f).

- Revisiting the *inference* of eaDCA models in an attempt to improve their efficiency, stability and representativity of specific RNA families;
- Assessing, in a more systematic manner, the capacity of eaDCA to predict key structural features at both the secondary and tertiary structure level, but also identify negative evolutionary pressure and, possibly, alternative structures.

## 2 Context and problems statement

*Direct Coupling Analysis (DCA)* is a statistical modeling paradigm for molecular families. A DCA *model* (also known as Potts models, or binary Markov networks) is based on a multiple sequence alignment of the family it attempts to model. It infers an energy function (ie. a weighting) for the space of aligned sequences $\{A, C, G, U, -\}^N$ where $\Sigma = \{A, C, G, U, -\}$ is the alphabet, $-$ denotes insertions/deletions and $N \in \mathbb{N}$ is the length of the alignment. Ideally, a *good* DCA model assigns low energies to sequences that share a common function/structure as the family, and high energies to non-functional sequences.

The energy function $E(w)$ is required to have the following form

$$-E(w \in \Sigma^N) = \sum_{1 \le i \le N} h_i(w_i) + \sum_{1 \le i < j \le N} J_{i,j}(w_i, w_j).$$

where $w_k$ denotes the k-th letter of the sequence $w$. The functions $(h_i)_{1 \le i \le N}$ are known as *local fields*, and $(J_{i,j})_{1 \le i < j \le N}$ as *couplings*. In the worst case scenario, there could be as much as $\Theta(N^2)$ couplings yet, in practice, few of them are non-zero. Non-zero couplings are known as *activated* couplings, and we refer to models with few activated couplings as *sparse*.

Two fundamental, mutually dependent, tasks in DCA modeling are:

- *Learning* a model from an alignment, by choosing the local fields $h_i$ and the couplings $J_{i,j}$ so as to: i) reproduce nucleotide distributions for individual sites and pairs thereof; and ii) maximize the joint emission of sequences in the training set;
- *Sampling* from the *Boltzmann* distribution induced by the inferred energy model:

$$P(w) = \frac{e^{-E(w)}}{Z(w)}$$

with

$$Z = \sum_{w \in \Sigma^N} e^{-E(w)}$$

a normalizing constant known as the *partition function*.

This distribution assigns high (resp. low) probability to low-energy (resp. high-energy) sequences, and can be used to design RNAs with similar function as those of the model family [4, 3]. Unfortunately, computing the partition function under general couplings is #P-hard, as DCA couplings can be set to emulate Ising models, for which computing the partition function is #P-Complete [9].

## The edge-activated DCA (eaDCA) framework

*Edge-activated Direct Coupling Analysis (eaDCA)*, introduced by Calvanese *et al.* [3], is a recent installment of DCA, which focuses on learning sparse models in a greedy manner. It adheres to the well-established framework of *maximum likelihood learning* used in statistical learning. In this framework, the likelihood function is

$$L(M; \text{dataset}) = \prod_{x \in \text{dataset}} P_M(x) = \prod_{x \in \text{dataset}} \frac{e^{-E(x)}}{Z}$$

where $P_M(x)$ is the probability of $x$ in the model $M = (J, h)$. It represents a measure of how well the model fits the data.

Within the *eaDCA* framework, the inference starts with an empty model, with no activated coupling, and using local fields $\{h_i\}_{1 \leq i \leq N}$ chosen to mimic site statistics. It then iteratively activates new couplings in a greedy manner, choosing at each step the coupling (and associated functions values) which maximizes the likelihood of the model. Fig. 2 details the criteria used to select new couplings along with their values, and when to stop the iteration.

Using this approach, its authors were able to produce sparse, yet accurate, models on a selection of RNA families [20]. Moreover, those results were obtained while activating a limited proportion of couplings, typically $\Theta(N)$ out of $\Theta(N^2)$, which turned out to substantially overlap with base pairs occurring in the structural consensus of the family.

The sparsity shown by eaDCA models is desirable in that it enables *interpretability* and *knowledge discovery*. A simple example of interpretable parameters is that of couplings $J_{i,j}(x, y)$ assigning large negative values when $x$ and $y$ are complementary nucleotides, and big positive values otherwise. Such couplings promote the pairing of columns $i$ and $j$, and suggest the existence of a corresponding evolutionary pressure towards maintaining a base pair in the column. Such couplings frequently involve pairs of columns that are backbone-adjacent, part of the consensus structure or, more generally, proximate in 3D models [3].

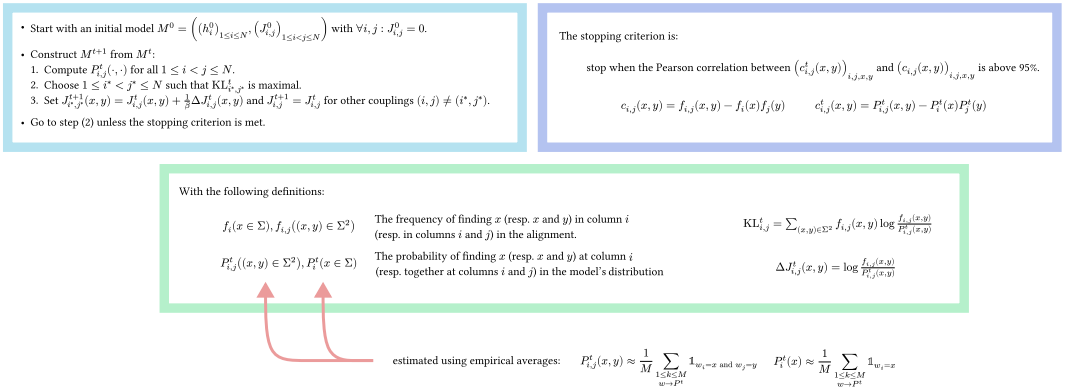



**Figure 2** Greedy maximum-likelihood learning in the style of eaDCA [3].

Back to learning, the original implementation of eaDCA heavily relies on *Markov Chain Monte Carlo (MCMC)* to approximately sample the distributions of intermediate models. These samples are used to compute estimates of marginals via empirical averages, a key step in the main loop of the inference algorithm (Figure 2). Such an approach represents an important shortcoming of the method: since the implementation of MCMC as it is used does not provide convergence bounds, approximate sampling is done with an unknown approximation error, and it follows that the precision of frequency estimates cannot be guaranteed. Moreover, the mixing time of the underlying Markov Chain, exponential in a worst-case analysis, is practically too long so the original designer of eaDCA ended up intentionally rely on out-of-equilibrium statistics whenever consequential couplings are added. Those limitations impede stability, and possibly reproducibility since the method is inherently random, the returned model structure depends on the seed provided to the random number generator.

## 3 Treewidth as a stronger and tractable notion of sparsity

The sparsity of eaDCA models suggests the use of exact algorithmic approaches to improve the computation. Namely, we now set out to show that the greedy approach can be implemented *exactly* with reasonable time complexity under sparsity assumption. By *exact*, we mean that the quantities of Figure 2 are computed with no other error except from those arising from floating-point operations, we don't mean that the learning procedure finds the maximum-likelihood model, indeed the greedy nature of the algorithm doesn't guarantee convergence to the maximum.

Towards that goal, we set out to design parameterized algorithms, based on the notion of *treewidth* of a graph, to solve the problem. The treewidth is a structural graph parameter which intuitively measures how *tree-like* a graph is, achieving low values for graph reduced to paths and trees, and larger $\Theta(n)$ values for graphs that are (near) complete. While formally divergent from the notion of sparsity, the treewidth is increasing upon adding edges, and typically correlates with the notion of sparsity. Graphs with small treewidth have a structure that can be exploited algorithmically, enabling an exact resolution of many NP-hard problems in time that grow polynomially on the size $N$ of the instance, and only super-polynomially (*e.g.* exponentially) on the treewidth $tw$ of the input graph. Such *Fixed-Parameter Tractable (FPT)* problems include Independent-Set and 3-SAT, both of which can be solved in time $O(\alpha^{tw} \cdot P(N))$ with $\alpha$ a constant and $P$ a polynomial.

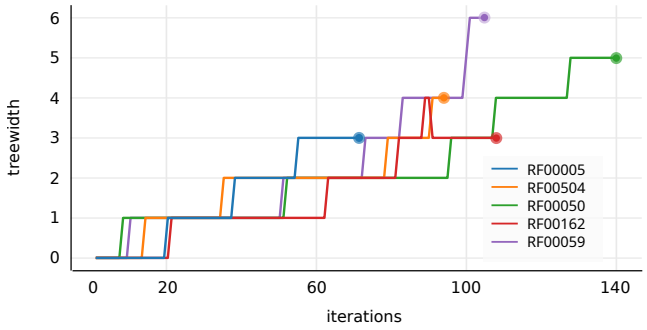



**Figure 3 Empirical treewidths of the eaDCA models produced by Calvanese *et al.* [3].** The learning procedure iterates a greedy strategy which stops when 95% of the single site and pair statistics are explained by the model. For all analyzed multiple sequence alignment, each associated with an RFAM families, the treewidth $tw$ of the final model remained modest (3 to 6), making $\mathcal{O}(n.|\Sigma|^{tw})$ DP-based FPT algorithms an attractive alternative to the unstable numerical procedures (MCMC) initially proposed by its authors.

As shown in Fig. 3, we observe that the graph of activated couplings, induced by eaDCA models learned by greedy maximum likelihood, have systematically low treewidth. This motivates the design of tractable and exact algorithms, as follows:

▶ **Theorem 1** (Parameterized greedy learning). *Given an alignment of $n$ sequences of length $N$, the maximum-likelihood greedy DCA model $M^T$ after $T$ steps can be obtained in time $O(nN^2 + tw^2 5^{tw} N^2 T + TC(tw; N))$ where:*

- *$tw$ is the treewidth of the graph of activated couplings of the model $M^T$;*
- *$C(tw; N)$ is the complexity of computing tree decompositions for graphs of treewidth $tw$.*

▶ **Theorem 2** (Parameterized sampling). *Given a DCA model $M$ of length $N$ along with a tree decomposition of the graph of its activated coulplings of width $tw$, we can draw a sample from its Boltzmann distribution in time $O(tw^2 5^{tw} N)$.*

It follows from the celebrated results of Bodlaender [2] that $C(tw; N) \in \mathcal{O}(N.2^{\mathcal{O}(tw^3)})$, therefore eaDCA learning is FPT for the treewidth. In practice, the cost $C$ of computing tree decompositions is negligible compared to the other time complexities.

▶ **Corollary 3.** *Greedy maximum-likelihood learning and sampling of DCA models are Fixed-Parameter Tractable for the treewidth of those models.*

Those algorithms are all based on the use of tree decompositions, that we now define:

▶ **Definition 4** (Tree decomposition). *For an undirected graph $G = (V, E)$, a tree decomposition of $G$ is a tuple $\mathcal{T} = (T, (X_v)_{v \in V(T)})$ where $T$ is a directed tree and $(X_v)_{v \in V(T)}$ is a family of subsets of $V(G)$ indexed by nodes of $V(T)$. The elements of $(X_v)_{v \in V(T)}$ are called the bags of the decomposition. Moreover, a decomposition must satisfy the following axioms:*

1. $\cup_{v \in V(T)} X_v = V(G)$ *(bags cover all the graph's vertices)*
2. $\forall (i, j) \in E(G), \exists v \in V(T) : i \in X_v$ *and* $j \in X_v$ *(every edge of the graph is covered by a bag)*
3. *if $v, w \in V(T)$ are two nodes of the decomposition and $x \in V(G)$ is a node of the graph such that $x \in X_v, x \in X_w$ then any node $u \in V(T)$ on the path between $v$ and $w$ must be such that $x \in X_u$. (running intersection property)*

*Finally, the* width *of the decomposition $\mathcal{T}$ is defined as $max_{v \in V(T)} |X_v| - 1$.*

The treewidth of a graph is the width of its decomposition of smallest width, thus a tree decomposition of width $tw$ certifies that its associated graph has treewidth at most $tw$. It gives an upper bound on how fast dynamic programming algorithms using this decomposition will run.

Although it is NP-Hard to compute the treewidth, or a tree decomposition of smallest width for a general graph [1], we can assume, as argued before, that we can easily obtain decompositions of small width for the graphs we are interested in. It is indeed what we observe in practice.

Moreover, we will also assume that those decompositions have size $O(N)$. This fact is guaranteed by our choice of heuristic used to compute them, but in general we can reduce the size of any bigger decomposition to at most $3N = O(N)$ vertices for $T$ [13].

## 3.1 Computing the partition function of DCA models

Now that we have presented the framework of tree decompositions, we show how it can be used to compute the *partition function* of DCA models. The partition function is not directly useful for our purpose, but is an important quantity in statistical physics, and once we have access to it, we can perform the useful tasks of learning or sampling of the models.

For a DCA model $M = ((h_i)_{1\leq i\leq N}, (J_{i,j})_{1\leq i<j\leq N})$, the graph induced by its activated couplings is $G_M = (\{1,..,N\}, E(G_M))$ where $\{i,j\} \in E(G_M) \iff J_{i,j}(\cdot,\cdot) \neq 0$. In the rest of the paper, we will assume that we have access to $\mathcal{T} = (T, (X_v)_{v\in V(T)})$ a tree decomposition of $G_M$ of low treewidth, along with a partition $f : V(T) \to \{J_{i,j}\}$ of the activated couplings of $M$:

$$\{J_{i,j} \neq 0\} = \bigsqcup_{v\in V(T)} f(v)$$

Such a partition is guaranteed to exist by the second axiom of tree decompositions (see definition 4), and can be found efficiently in time $O(tw^2 N)$. Also for the sake of concision, we will ignore the local fields $(h_i)_{1\leq i\leq N}$ in our algorithmic treatment. Those can be handled by extending the theory with loops $J_{i,i} = h_i$.

Moreover, we will use the following notations:

- $D_v \subset V(T)$ are the descendants of node $v$ (included), in $T$.
- $Y_v = \cup_{u\in D_v} X_u$
- $X_{pa(v)} \subset \{1,..,N\}$ is the bag $X_w$ for $w$ the parent of $v$ if it exists or $\emptyset$ if $v$ has no parent.
- For $s \in \Sigma^A, s' \in \Sigma^B$ with $A \subset B$, $s \subset s'$ means that $s'$ agrees with $s$ on its domain.

We can define the partition function:

▶ **Definition 5** ((generalized) partition function).
*Let $v \in V(T)$ be a node of $\mathcal{T}$, the partition function $Z_v$ is the function:*

$$Z_v(s \in \Sigma^{X_v \cap X_{pa(v)}}) = \sum_{\substack{s'\in\Sigma^{Y_v} \\ s\subset s'}} e^{-\sum_{u\in D_v}\sum_{J_{i,j}\in f(u)} J_{i,j}(s(i),s(j)))} = \sum_{\substack{s'\in\Sigma^{Y_v} \\ s\subset s'}} e^{-E_v(s)}$$

*with $E_v(s) = \sum_{u\in D_v}\sum_{J_{i,j}\in f(u)} J_{i,j}(s(i), s(j))$.*

Note that when $v$ is the root we recover $Z_v(\emptyset) = Z = \sum_{s\in\Sigma^N} e^{-\sum_{J_{i,j}} J_{i,j}(s_i,s_j)}$ because $D_v = V(T)$ and $f$ is a partition of $M$'s couplings. For a general $v \in V(T)$, the quantity $Z_v(s)$ can be interpreted as the weight of the assignments $s'$ that extend $s$ in the Boltzmann distribution, when considering the submodel composed of columns/couplings below $v$ in $\mathcal{T}$.

The key lemma that justifies the use of tree decompositions is the following. It provides a recursive definition of partition functions that can be used as a dynamic programming equation:

▶ **Lemma 6** (DP equation for the partition function). *Let $v \in V(T)$ be a vertex of the decomposition such that: $u_1, .., u_k$ are the direct children of $v$ in $T$. Then the following holds:*

$$Z_v(s \in \Sigma^{X_v \cap X_{pa(v)}}) = \sum_{\substack{s' \in \Sigma^{X_v} \\ s \subset s'}} e^{-\sum_{J_{i,j} \in f^{-1}(v)} J_{i,j}(s'(i), s'(j))} \times \prod_{1 \leq i \leq k} Z_{u_i}(s'|_{X_{u_i} \cap X_v})$$

*where $s|_S$ is the restriction of $s$ onto the columns of $S$.*

**Proof.** (available in the appendix) ◀

A dynamic programming algorithm implementing this formula has time complexity:

$$O\left(\sum_{v \in V(T)} |X_v \cap X_{pa(v)}| \times |X_v - X_{pa(v)}| \times |f^{-1}(v)| \times deg(v)\right) = O(5^{tw+1} tw^2 N)$$

## 3.2 Sampling from the Boltzmann distribution of a DCA model

Access to the partition functions enables us to sample the model directly through the following algorithm:

▶ **Lemma 7.** *Let $v_1, .., v_n$ be a reversed topological ordering of $T$ starting from the root. The following algorithm samples from the Boltzmann distribution of $M$:*

*Let $s$ be the unique assignment of $\Sigma^{\emptyset}$.*
***for*** *$1 \leq i \leq n$* ***do***
  *Let $B = X_{v_i} - dom(s)$.*
  *Sample $s' \in \Sigma^B$ with probability proportional to $Z_{v_i}((s + s')_{|X_{v_i} \cap X_{pa(v_i)}})$.*
  *Extend $s$ with the values of $s'$.*
***end for***

**Proof.** (available in the appendix) ◀

The sample $s$ is built from a partial assignment of $\Sigma^{\{1,..,N\}}$ whose domain is gradually populated with the variables present in the sets $X_v$ of the nodes $v \in V(T)$ that are visited. The intermediate samples are drawn from a distribution whose values depend entirely on $Z_v$ and $s$, showcasing that the hard part of this sampling algorithm is the computation of partition functions.

## 3.3 Computing pairwise marginals of the Boltzmann distribution

The marginals $P_{i,j}(x, y)$ for $1 \leq i < j \leq N$ and $x, y \in \Sigma$ are central to the greedy learning algorithm. In a similar way to the original eaDCA paper [3], we may estimate those marginals by using samples from the Boltzmann distribution (Lemma 7), this time with the guarantee that the sampling is not approximate. But it is in fact possible to remove the randomness entirely, using the technique that we now describe.

To compute the marginals, we need to have access to an additional type of partition functions known as an outside partition function:

▶ **Definition 8** (outside partition function). *Let $v \in V(T)$ be a node of $\mathcal{T}$, the outside partition function $Z_v^o$ is the function:*

$$Z_v^o(s \in \Sigma^{X_v \cap X_{pa(v)}}) = \sum_{\substack{s' \in \Sigma^{Y_v} \\ s \subset s'}} e^{-\sum_{u \in V(T)-D_v} \sum_{J_{i,j} \in f(u)} J_{i,j}(s'(i),s'(j)))} = \sum_{\substack{s' \in \Sigma^{Y_v} \\ s \subset s'}} e^{-E_v^o('s)}$$

*with $E_v^o('s) = \sum_{u \in V(T)-D_v} \sum_{J_{i,j} \in f(u)} J_{i,j}(s'(i), s'(j))$. The only change here is that the sum is done over $u \in V(T) - D_v$.*

Similarly, outside partition functions can be computed in the same time complexity using the following dynamic programming equation:

▶ **Lemma 9.** *Let $v \in V(T)$ be a node of the decomposition such that $u_1, .., u_k$ are the direct children of $v$ in $T$. Then the following holds $\forall 1 \leq l \leq k$:*

$$Z_{u_l}^o(s \in \Sigma^{X_{u_l} \cap X_v}) = \sum_{\substack{s' \in \Sigma^{X_v} \\ s \subset s'}} e^{-\sum_{J_{i,j} \in f^{-1}(v)} J_{i,j}(s'(i),s'(j))} \times Z_v^o(s'_{|X_v \cap X_{pa(v)}}) \times \left( \prod_{\substack{1 \leq i \leq k \\ i \neq l}} Z_{u_i}(s'|_{X_{u_i} \cap X_v}) \right)$$

**Proof.** (omitted, similar proof to Lemma 6: decompose the non-descendants of $u_l$ as its parent $v$, its siblings' descendants, and its parent's non-descendants.) ◀

The main differences here being that the computation is performed starting from the top of the tree $T$ towards the leaves, and that both types of partition functions are used.

It can be observed that $E_v(s) + E_v^o(s) = E(s)$ because $V(T) = D_v \sqcup (V(T) - D_v)$ suggesting that the two types of partition functions decompose the model into two parts. Due to the separation properties of tree decompositions (see Lemma 12 in the appendix) this intuition extends further with the following lemma:

▶ **Lemma 10.** *Let $v \in V(T)$ and $P_v$ be the marginal of the Boltzmann distribution restricted to the columns of $X_v \cap X_{pa(v)}$. Then for $s \in \Sigma^{X_v \cap X_{pa(v)}}$ we have that:*

$$Z_v^o(s) \times Z_v(s) = \sum_{\substack{s' \in \Sigma^{\{1,..,N\}} \\ s \subset s'}} e^{-\beta E(s')} \propto P_v(s)$$

**Proof.** (available in the appendix) ◀

This provides us with a way to compute some of the marginals of the Boltzmann distribution. Namely, those restricted to $X_v \cap X_{pa(v)}$ for some $v \in V(T)$. If a pair of columns $\{i, j\}$ is included in one of them then we can compute $P_{i,j}$ by summing out the other variables. Indeed, letting $A \subset B \subset \{1, .., N\}$ we have from probability theory that for any distribution $Q$: $Q(s \in \Sigma^A) = \sum_{\substack{s' \in \Sigma^B \\ s \subset s}} Q(s')$. However, for a fixed tree decomposition there can only be at most $O(tw^2 N)$ such pairs $\{i, j\}$ while we need to compute $\Omega(N^2)$ of them. To circumvent this problem, we can construct decompositions $\mathcal{T}^{i^*}$ for $1 \leq i^* \leq N$ from $\mathcal{T}$ such that all the edges in $(\{i^*, j\})_{1 \leq j \leq N}$ are such that there exists $v \in V(T^{i^*})$ with $\{i^*, j\} \subset X_v \cap X_{pa(v)}$.

**Algorithm 1** Compute pairwise marginals of a DCA model given a tree decomposition $\mathcal{T}$.

**for** $1 \leq i \leq N$ **do**
  Compute $\mathcal{T}^i$ from $\mathcal{T}$ in time $O(N(tw+1))$.
  Compute $Z_v$ for every $v \in V(T^i)$ in time $O(5^{tw+1}tw^2N)$.
  Compute $Z_v^o$ for every $v \in V(T^i)$ in time $O(5^{tw+1}tw^2N)$.
  **for** $v \in V(T^i)$ **do**
    Let $P_v(s) = Z_v(s) \times Z_v^o(s)$.
    **for** $\{i,j\} \subset X_v$ with $i < j$ **do**
      **if** $P_{i,j}$ has not been computed yet **then**
        Compute $P_{i,j}$ from $P_v$ by summing out $X_v - \{i,j\}$.
      **end if**
    **end for**
  **end for**
**end for**

▶ **Lemma 11** (Augmented tree decompositions). *Let $\mathcal{T}$ be a tree decomposition of $G$ of size $s \in \mathbb{N}$ and $x \in V(G)$. We can build a decomposition $\mathcal{T}^x$ of size $2s$ and treewidth at most $tw(\mathcal{T})+1$ such that for every $y \in V(G)$ there exists $v_y \in V(T^x)$ such that:*

$$\{x,y\} \subset X_{v_y} \cap X_{pa(v_y)}$$

**Proof.** (available in the appendix) ◀

Using this lemma and the rest of the techniques that we have introduced previously, we can compute the marginals $P_{i,j}$ for all $1 \leq i < j \leq N$ as shown if Algorithm 1 in time $O\left(5^{tw+2}N^2tw^2\right)$.

This completes the proof of Theorem 1 if we use the marginals $P_{i,j}$ to implement $T$ steps of eaDCA as in Figure 2, and we take into account the pre-computation of the statistics $f_{i,j}, f_i$ in time $O(nN^2)$ as well as the cost of finding tree decompositions.

## 4 Empirical analyses

The ability of eaDCA to effectively model RNA families was already investigated in the original paper [3] on a limited selection of families associated with large alignments. However, our FPT alternative enables the assessment of the method at a larger scale, and test various hypotheses on the behavior of the framework. To that purpose, we implemented our exact treewidth-based revisiting of eaDCA as a `Rust` software named `eaDCA*`. The code, data and detailed per-family results associated with these experiments are available at `https://codeberg.org/samuel-gardelle/wabi26-eaDCAstar`.

**Datasets.** We consider family-level multiple sequence alignment, downloaded from the 15.1 version of RFAM [20]. We used the *seed alignments* of families, usually established through an expert-driven combination of computational methods and manual curation.

Out of the 4 227 RFAM families currently in RFAM, 41 were selected (see full list in Appendix Table A.1) to match the following criteria:

1. The (horizontal) alignment length must be between 100 and 400 nts;
2. The alignment should include at least 100 sequences;
3. The average pairwise Hamming distance should not exceed 80%;
4. The consensus structure of the family should be tagged as "Published" (*vs* "Predicted");
5. At least 50% of positions should be paired.

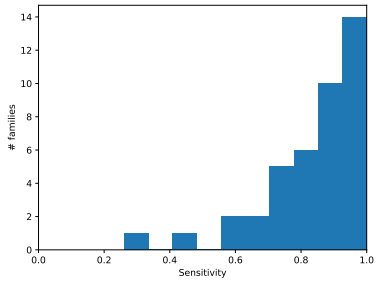



**Figure 4** Distribution of sensitivity, *i.e.* proportion of base pairs from the RFAM consensus structure that are activated as couplings by `eaDCA`$^\star$.

The length requirement (Criterion 1) is mainly meant to: i) avoid smaller RNA families, such as the 1 500+ miRNA precursors associated with seed alignments of lower quality; as well as ii) larger alignments resulting from horizontal expansion (gaps), indicating lower alignment quality while inducing unreasonable computational demands. Criterion 2 and 3 represent minimal prerequisites for the alignment (and underlying RNA family) to showcase some level of sequence-level covariation. We require some level of expert scrutiny over the consensus structural model (Criterion 4). Finally, we forced the presence of a critical proportion of base pairs (Criterion 5), hopefully indicative of a family undergoing evolutionary pressure at a structural level.

**Experiments.** We conducted further computational experiments, by systematically learning models for the families of the RFAM database. We used the alignments during the learning phase to compute the frequencies $f_{i,j}(\cdot,\cdot)$ and $f_i(\cdot)$ using a pseudocount estimator ($\alpha = 1$). For each family, we ran `eaDCA`$^\star$ to infer $T = 200$ couplings, prematurely stopping whenever the graph induced by the current couplings attains a treewidth exceeding 5. The evolution of the treewidth is depicted in Figure 9 in appendix.

Note that couplings are selected based on coevolutionary metrics alone and, in particular, do not necessarily reflect the capacity of two columns to form base pairs. In order to better distinguish couplings that are consistent with base pairing rules, we define the *Pairwise-Compatibility (PC)* score as

$$PC(i,j) = \frac{\sum_{\{x,y\}\in\{\{G,C\},\{U,A\},\{G,U\}\}} f_{i,j}(x,y)}{\sum_{\substack{x,y\in\Sigma \\ x\neq - \text{ and } y\neq -}} f_{i,j}(x,y)}.$$

For two unrelated, independent columns in an alignment, with no gaps the PC is expected to be $\frac{6}{16} = 0.375$.

The results, detailed henceforth, show that the method is applicable to a wider range of families. Indeed, we confirmed that `eaDCA`$^\star$ is able to recover most basepairs present in the RFAM consensus structure. Moreover, the presence of couplings in the model correlate highly with small distances in PDB structures. Both of these points show the ability of DCA models to highlight positive evolutionary pressures, such that the presence of basepairs. We were able to go further, and show that some couplings suggest the presence of negative evolutionary pressures preventing e.g. stack extension, helix recombination, or the formation of additional structures in unpaired regions.

## 4.1 Testing the capacity of eaDCA$^\star$ to recover RFAM consensus structures

We executed eaDCA$^\star$ on the 41 RFAM families as described above on a domestic laptop, and obtained the couplings in less than a minute for each of the families. In every case, in the first 10 seconds we already obtained 100 couplings and a treewidth $tw \leq 5$. Details on the final number of couplings and the final treewidth can be found in the appendix.

Firstly, for a given RFAM family, we assess the capacity of eaDCA$^\star$ to recover the set $B$ of base pairs found in the consensus structure. Towards that goal, we compute the *sensitivity* of a model $M$, activating a list $E_M$ of couplings, as $\text{Sens}(M) := \frac{|B \cap E_M|}{|B|}$.

As can be seen in Fig. 4, which shows the distribution of the sensitivity metrics, for 27 out of the 41 families, the models inferred by eaDCA$^\star$ achieve sensitivities of 80% or more, and only 2 models feature less than 50% of base pairs in the RFAM consensus. This is reassuring, yet not entirely surprising since consensus structures are usually built from a specific type of co-evolutionary signal (covariance models [5]). Still, such performances are remarkable as, for many families, the number of base pairs in the consensus is close to the number of activated couplings. Moreover, consensus structures are usually drafted from a combination of evolutionary and physics-based evidence [7], enabling a prediction of base pairs that do not exhibit any meaningful coevolution (*e.g.* over fully conserved columns). Overall, we observe that eaDCA is generally able to prioritize couplings associated with compensatory mutations.

We illustrate in Fig. 5 the behavior of eaDCA$^\star$ on six hand-picked RNA families. A visual inspection suggests that eaDCA$^\star$ is able to recover most of the consensus base pairs, usually very early across iterations. Notably, the method does not appear to be hindered by the presence of crossing interactions/pseudoknots (see RF02996). We observe negative couplings, activated but associated with incompatible pairs of nucleotides (low PC value/red), are often found at the ends of helices, seemingly preventing their extension (see Subsection 4.2). Negative coupling can also be found within single regions that are predicted to remain unpaired (see RF00020), or across unpaired regions, seemingly as the result of an evolutionary pressure against their interaction (see RF02944 and RF02968).

Note that typical structural analyses consider some measure of specificity, such as the PPV, in addition to sensitivity. However, such an assessment is delicate as detecting structurally-conserved base pairs is not the sole goal of eaDCA. Indeed, it is known that activated couplings also frequently include neighboring positions on the sequence and 3D levels, as well as evidences, discussed below, of a negative selective pressure, acting against structurally-deleterious mutations. Moreover, any reasonable assessment of specificity would require a well-calibrated stopping criterion, an aspect that leave to future work.

## 4.2 Revealing negative evolutionary pressure

We further analyzed couplings involving incompatible columns, in an attempt to reveal negative evolutionary signals in the couplings that were inferred in the previous experiment. Towards this goal, we refine the notion of *disruptive base pairs* introduced by Yao *et al.* [23], and classify the couplings of eaDCA$^\star$ models depending on their position in the consensus structure.

We distinguish 3 cases that are believed to correspond to negative evolutionary constraints:

- *Stack extensions* are couplings whose positions are present directly above or below an helix of the consensus structure. We also transitively include in that definition couplings that are below/above other stack extensions. Depending on the compatibility of columns, measured by the PC metrics, we posit that such couplings reflect a selective pressure against the further extension of an helix;

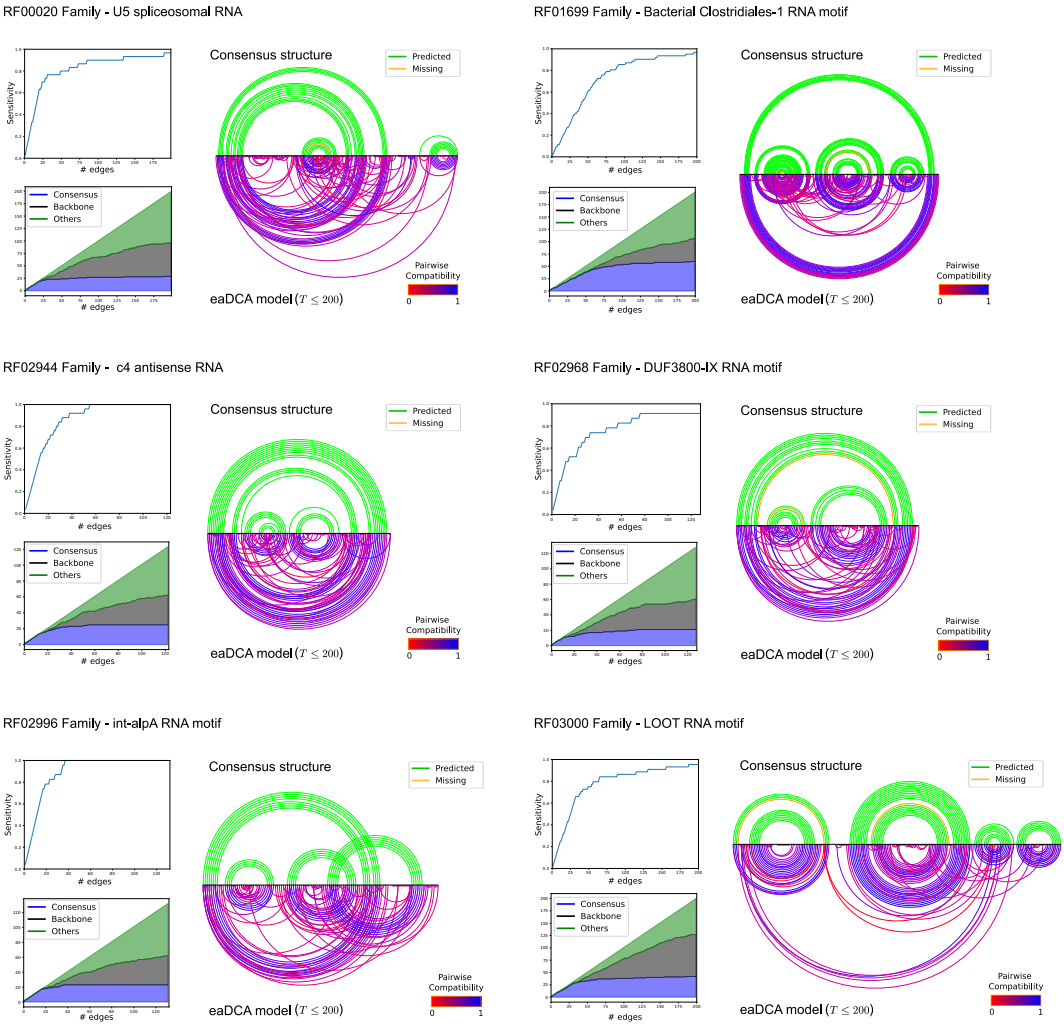



**Figure 5** RFAM consensus structures and typical eaDCA models. For each RFAM family, we report the base pairs contained in the consensus structure (top-right) and those inferred by the couplings (bottom-right). Consensus base pairs are color-coded to illustrate their presence in (green) or absence from (orange) the consensus, and eaDCA couplings are drawn using a gradient from red to blue to indicate the proportion of compatible base pairs found in the alignment. The evolution, across iterations, of the sensitivity – the proportion of the consensus BPs found in the eaDCA model – is represented in the top left. Finally, we report (bottom-left), a breakdown of couplings into base pairs that are in the family consensus (blue), proximate in the backbone (black) or others (green).

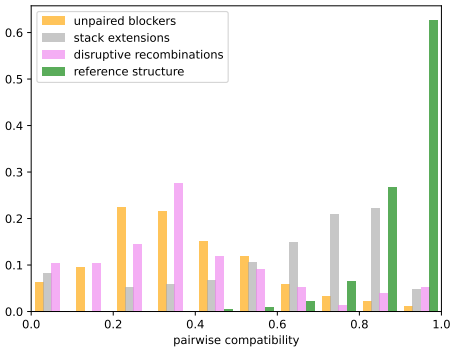


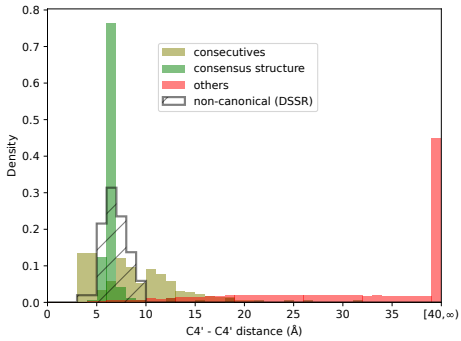

**Figure 6** Distribution of pairwise compatibility (PC) among couplings, stratified based on their relationship with the consensus structure. Unpaired blocked couplings are contained within an unpaired region; stack extensions represent possible base pairs that would stack onto an existing helix; disruptive recombinations mediate (or forbid) interaction between two distinct helices.

- *Unpaired blockers* are couplings that involve two positions present in the same stretch of unpaired positions in the consensus structure. We posit that their presence reflects a loss of fitness incurred by the formation of additional helices, in a region whose accessibility is functionally important;
- *Disruptive recombinations* are couplings whose positions are present in the unpaired regions of two distinct hairpins. We posit that such couplings indicate an evolutionary pressure against the formation of a competing structure including the coupling.

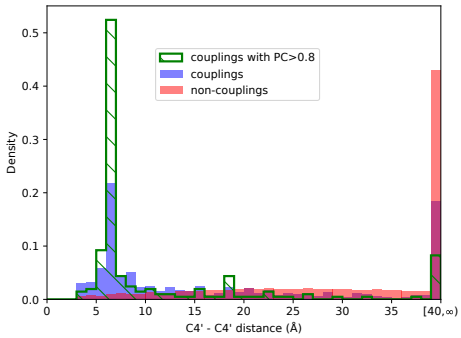
All of these mechanisms require that the associated couplings favor non-complementary base pairs to hinder the formation of base pairs. In other words, we expect the associated couplings to have low PC score. Figure 6 reports the distribution of PC values for the 3 categories of negative evolutionary couplings inferred by `eaDCA`$^\star$. For reference, we also report the PC distribution for couplings occurring in the consensus structure.

As expected, a clear divergence is observed between the consensus pairs and the negative couplings, the latter generally achieving much lower compatibility with respect to the PC metrics. An exception to that trend is observed for stack extensions. This is not overly surprising, since consensus structures within RFAM are predicted using diverse (at least partly manual) methodologies, some of which can be overly conservative especially within columns featuring gaps. We believe that such base pairs, having good mutual information, compatibility, and stacking onto an existing helix, could probably be safely added to the RFAM consensus structure.

Unpaired blockers and disruptive recombinations appear to induce PC distributions, with a low median value around 0.3. This moderate value is interesting, as it does not prevent the formation of base pairs in every sequence. It suggests that the repression of helices in unpaired regions may induce pervasive selection of limited intensity, inducing disjunctive incompatibility constraints, in a non-targeted manner and in combination with other local constraints. A notable difference can however be observed for extreme PC values (close to 1), where unpaired blockers are virtually absent, while 6% of disruptive recombinations achieve a PC close to 1. We hypothesize that this behavior may reflect the existence of alternative structures which would be important, and whose base pairs would therefore be positively selected by evolution.




**(a)** Distance distribution for pairs of columns in RFAM MSA: (Green) Base pairs found in the RFAM consensus structure; (Gold) Pairs $(i, j)$ of quasi-consecutive positions ($j \leq i + 3$); (White/-Dashed) Pairs annotated as non-canonical base pairs in at least one 3D structure; (Red) Other columns pairs (background).




**(b)** Distance profile among `eaDCA`$^\star$-inferred couplings (Blue) versus non-couplings (Red). We additionally report the distances over the subset of `eaDCA`$^\star$ couplings associated with good compatibility score ($PC > 0.8$).

**Figure 7** Distance distribution of 4'-4' atomic distances obtained from PDB structures projected onto the RFAM reference alignments. Distances of more than 40Å are aggregated in the final bin.

## 4.3 Geometric distances of eaDCA couplings

We further estimate the ability of eaDCA to identify informative couplings towards 3D modeling, We consider tertiary structure data available in the RFAM alignments into our analysis. Among the 41 families selected previously, we were able to obtain high-quality 3D models, and alignments to their multiple sequence alignments within RFAM [20], for 6 of them (see Appendix Note A.2). We retrieved the corresponding 42 single chains from the PDB database [22], and base pair annotations produced using the DSSR [17] software suite, from the NAKD database [16]. For all 3D models, we computed the geometric distance between all pairs of 4' atoms (backbone) in the alignment, and report in Figure 7 the distances stratified by presence/absence in the consensus and inferred eaDCA model.

Figure 7a mainly represents a validation of the alignments of 3D models to RFAM family. As expected, for base pairs in the RFAM consensus, the distribution of 4'-4' distance is extremely concentrated around 6.5 Å, in agreement with textbook knowledge in structural biology. The distance distribution for consecutive nucleotides is more widespread, but consistent with the expected distance between nucleotides separated by at most 3 nucleotides. Finally, pairs of columns, associated with non-canonical base pairs in at least one aligned PDB structure, induce 4'-4' distances that are similar on average as consensus pairs, with higher yet reasonable variance. All of those distributions strongly differ from that of other pairs, which virtually always induce distances of more than 40Å. This confirms the quality of the structural alignments found in RFAM, and their potential utility in assessing couplings.

The distance distribution induced by couplings is more informative, and provides promising directions towards future predictive methods. As shown in Figures 7b and 8, the 3D 4'-4' distance induced by `eaDCA`$^\star$-produced couplings strongly differs from the non-couplings (background) distribution. The distance distribution for couplings is largely concentrated around the value of 6.5Å, as is expected for base pairs. Couplings also feature a minority of larger distances, inconsistent with direct base pairing, with some associated with distances greater than 40Å.

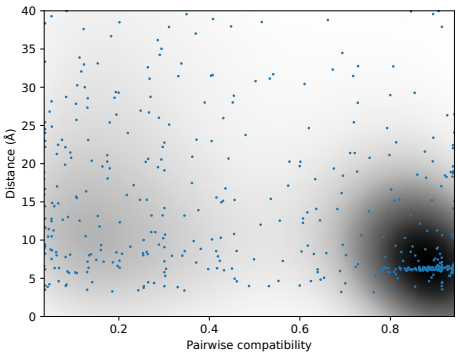



**Figure 8** 2D scatter plot of the distribution of coupling distances and their pairwise compatibility superimposed with an estimate of the density. Couplings with a distance above 40Å are not shown.

Those more distant couplings may indicate alignment/annotation errors, or reflect negative evolutionary pressure, which does not necessarily involve positions in direct contact. The hypothesis of a negative selective pressure is also partly supported by the fact that restricting our analysis to compatible couplings ($PC > 0.8$) greatly increases the population of the *canonical* 6.5Å bin, while more than halving the proportion of couplings at distances beyond 40Å.

## 5 Conclusion

In the context of the statistical modeling of RNA families, we revised the algorithmic framework underlying eaDCA [3], a proven greedy approach with the advantage of producing sparse and interpretable models. We exploited their sparsity using the stronger notion of treewidth, to devise rigorous, stable, deterministic and fixed-parameter tractable alternative algorithms for the modeling tasks (learning, sampling). We further evaluated the ability of our implementation, dubbed `eaDCA`$^\star$, to infer couplings with structurally significant roles. We did so in a systematic manner, and when applicable, using the available 3D structures. Our analysis shows that Direct-Coupling Analysis can be used to mine co-evolutionary signals present in RNA families, and that it is able to recover couplings lists associated with low tree widths, where the base pairs of the consensus structure feature prominently. We further verify that `eaDCA`$^\star$ couplings are associated with low distances in PDB structures, and that for some of them that may show signs of negative evolutionary pressures. Our work suggests further investigations over the use and impact of non-greedy approaches for the choice of couplings (e.g. maximum coupling, Nussinov, local search), or the comparison with other standard approaches that make use of co-evolution [11].

## References

1 Stefan Arnborg, Derek G Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in ak-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987.

2 Hans L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '93, pages 226–234, New York, NY, USA, 1993. Association for Computing Machinery. `doi:10.1145/167088.167161`.

3 Francesco Calvanese, Camille N Lambert, Philippe Nghe, Francesco Zamponi, and Martin Weigt. Towards parsimonious generative modeling of RNA families. *Nucleic Acids Research*, 52(10):5465–5477, April 2024. doi:10.1093/nar/gkae289.

4 Eleonora De Leonardis, Benjamin Lutz, Sebastian Ratz, Simona Cocco, Rémi Monasson, Alexander Schug, and Martin Weigt. Direct-coupling analysis of nucleotide coevolution facilitates RNA secondary and tertiary structure prediction. *Nucleic Acids Research*, page gkv932, September 2015. doi:10.1093/nar/gkv932.

5 Sean R. Eddy. Homology searches for structural RNAs: from proof of principle to practical use. *RNA*, 21(4):605–607, March 2015. doi:10.1261/rna.050484.115.

6 Samuel Gardelle. eaDCAstar. Software, version 1., swhId: swh:1:dir:e53ad07a87e78c5d6e48b33986c10a1c937ed706 (visited on 2026-08-14). URL: https://codeberg.org/samuel-gardelle/wabi26-eaDCAstar, doi:10.4230/artifacts.27542.

7 Andreas Gruber, Ivo Hofacker, Stephan Bernhart, Peter Stadler, and Sebastian Will. RNAalifold: improved consensus structure prediction for RNA alignments. *BMC Bioinformatics*, 9(Art. 474), 2008. doi:10.1186/1471-2105-9-474.

8 Paul G. Higgs and Niles Lehman. The RNA world: molecular cooperation at the origins of life. *Nature Reviews Genetics*, 16(1):7–17, November 2014. doi:10.1038/nrg3841.

9 Mark Jerrum and Alistair Sinclair. Polynomial-time approximation algorithms for the Ising model. *SIAM Journal on Computing*, 22(5):1087–1116, 1993. doi:10.1137/0222066.

10 Ioanna Kalvari, Joanna Argasinska, Natalia Quinones-Olvera, Eric P Nawrocki, Elena Rivas, Sean R Eddy, Alex Bateman, Robert D Finn, and Anton I Petrov. Rfam 13.0: shifting to a genome-centric resource for non-coding RNA families. *Nucleic acids research*, 46:D335–D342, January 2018. doi:10.1093/nar/gkx1038.

11 Aayush Karan and Elena Rivas. All-at-once RNA folding with 3d motif prediction framed by evolutionary information. *Nature Methods*, 22(10):2094–2106, 2025.

12 Kenneth Katz, Oleg Shutov, Richard Lapoint, Michael Kimelman, Rodney Brister, and Christopher O'Sullivan. The sequence read archive: a decade more of explosive growth. *Nucleic Acids Research*, 50(D1):D387–D390, November 2021. doi:10.1093/nar/gkab1053.

13 Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

14 Camille N Lambert, Vaitea Opuu, Francesco Calvanese, Polina Pavlinova, Francesco Zamponi, Eric J Hayden, Martin Weigt, Matteo Smerlak, and Philippe Nghe. Exploring the space of self-reproducing ribozymes using generative models. *Nature communications*, 16(1):7836, 2025.

15 Marlen C. Lauffer, Willeke van Roon-Mom, and Annemieke Aartsma-Rus. Possibilities and limitations of antisense oligonucleotide therapies for the treatment of monogenic disorders. *Communications Medicine*, 4(1), January 2024. doi:10.1038/s43856-023-00419-1.

16 Catherine L Lawson, Helen M Berman, Li Chen, Brinda Vallat, and Craig L Zirbel. The nucleic acid knowledgebase: a new portal for 3d structural information about nucleic acids. *Nucleic Acids Research*, 52(D1):D245–D254, January 2024. doi:10.1093/nar/gkad957.

17 Xiang-Jun Lu, Harmen J Bussemaker, and Wilma K Olson. DSSR: an integrated software tool for dissecting the spatial structure of RNA. *Nucleic acids research*, 43(21):e142–e142, 2015.

18 Mihir Metkar, Christopher S. Pepin, and Melissa J. Moore. Tailor made: the art of therapeutic mRNA design. *Nature Reviews Drug Discovery*, 23(1):67–83, November 2023. doi:10.1038/s41573-023-00827-x.

19 Faruck Morcos, Andrea Pagnani, Bryan Lunt, Arianna Bertolino, Debora S. Marks, Chris Sander, Riccardo Zecchina, José N. Onuchic, Terence Hwa, and Martin Weigt. Direct-coupling analysis of residue coevolution captures native contacts across many protein families. *Proceedings of the National Academy of Sciences*, 108(49):E1293–E1301, 2011. doi:10.1073/pnas.1111471108.

20 Nancy Ontiveros-Palacios, Emma Cooke, Eric P Nawrocki, Sandra Triebel, Manja Marz, Elena Rivas, Sam Griffiths-Jones, Anton I Petrov, Alex Bateman, and Blake Sweeney. Rfam 15: RNA families database in 2025. *Nucleic Acids Research*, 53(D1):D258–D267, January 2025. doi:10.1093/nar/gkae1023.

21 The ENCODE Project Consortium. Identification and analysis of functional elements in 1% of the human genome by the ENCODE pilot project. *Nature*, 447(7146):799–816, June 2007. doi:10.1038/nature05874.

22 wwPDB consortium. Protein data bank: the single global archive for 3d macromolecular structure data. *Nucleic Acids Research*, 47(D1):D520–D528, January 2019. doi:10.1093/nar/gky949.

23 Hua-Ting Yao, Jérôme Waldispühl, Yann Ponty, and Sebastian Will. Taming disruptive base pairs to reconcile positive and negative structural design of rna. In *RECOMB 2021-25th international conference on research in computational molecular biology*, 2021.


# A Appendix

The proofs in this section rely on the following lemma:

▶ **Lemma 12** (Separation lemma). *Let $\mathcal{T}$ be a tree decomposition and $v \in V(T)$ one of its nodes. Let $C_1, .., C_k$ be the connected components obtained by removing $v$ from $T$. Let $\mathcal{X}_l = \bigcup_{v \in C_l} X_v$ be the nodes of $G$ involved in each component. Then we have that:*

$$X_v \sqcup \bigsqcup_{1 \leq l \leq k} \mathcal{X}_l - X_v = V(G)$$

**Proof.** Assume to the contrary that there exists $l_0, l_1, x \in V(G)$ such that $x \in (\mathcal{X}_{l_0} - X_v) \cap (\mathcal{X}_{l_1} - X_v)$ and $l_0 \neq l_1$. Let $v_0 \in C_{l_0}$ (resp. $v_1 \in C_{l_1}$) be such that $x \in X_{v_0}$ (resp. $x \in X_{v_1}$). Then since $T$ is a tree there is a unique path from $v_0$ to $v_1$, with $v$ on this path since it separated $C_{l_0}$ and $C_{l_1}$. Because $x \in X_{v_0} \cap X_{v_1}$ by the running intersection property (see Definition 4) we have that $x \in X_v$ which is a contradiction with $x \in \mathcal{X}_{l_0} - X_v$. ◀

▶ **Lemma 6** (DP equation for the partition function). *Let $v \in V(T)$ be a vertex of the decomposition such that: $u_1, .., u_k$ are the direct children of $v$ in $T$. Then the following holds:*

$$Z_v(s \in \Sigma^{X_v \cap X_{pa(v)}}) = \sum_{\substack{s' \in \Sigma^{X_v} \\ s \subset s'}} e^{-\sum_{J_{i,j} \in f^{-1}(v)} J_{i,j}(s'(i), s'(j))} \times \prod_{1 \leq i \leq k} Z_{u_i}(s'|_{X_{u_i} \cap X_v})$$

*where $s|_S$ is the restriction of $s$ onto the columns of $S$.*

**Proof.** Let $C_0, C_1, .., C_k$ be the connected components obtained by removing $v$ from $T$, such that:

- $C_0$ is the connected component containing the parent of $v$.
- $u_l \in C_l$ is the child of $v$ adjacent to $v$.

This decomposition of $T$ yields the following decomposition of the energy $E(w)$:

$$E(w) = \left( \sum_{1 \leq l \leq k} E_{v_l}(w) \right) + \left( \sum_{J_{i,j} \in f(v)} J_{i,j}(w_i, w_j) \right)$$

Moreover, by definition we have that $E_{v_l}(\cdot)$ only depends on $w_{|Y_{u_l}}$ since $Y_{u_l} = \mathcal{X}_l$ ($u_l$ is the root of $C_l$):

$$E(w) = \left( \sum_{1 \leq l \leq k} E_{v_l}(w_{|Y_{u_l}}) \right) + \left( \sum_{J_{i,j} \in f(v)} J_{i,j}(w_i, w_j) \right)$$

Use this to rewrite the definition of the partition function $Z_v(s \in \Sigma^{X_v \cap X_{pa(v)}})$:

$$Z_v(s) = \sum_{\substack{s' \in \Sigma^{Y_v} \\ s \subset s'}} e^{-E(s')} = \sum_{\substack{s' \in \Sigma^{Y_v} \\ s \subset s'}} \left( e^{-\sum_{J_{i,j} \in f(v)} J_{i,j}(s'(i), s'(j))} \right) \prod_{1 \leq l \leq k} e^{-E_{u_l}(s'_{|Y_{u_l}})}$$

By applying Lemma 12, we have that the sets $(Y_{u_l} - X_v)_l$ are disjoint, which allows us to decompose the choice of $s'$:

$$\begin{aligned} Z_v(s) &= \sum_{\substack{s' \in \Sigma^{X_v} \\ s \subset s'}} \left( e^{-\sum_{J_{i,j} \in f(v)} J_{i,j}(s'(i), s'(j))} \right) \prod_{1 \leq l \leq k} \sum_{\substack{s_l \in \Sigma^{Y_{u_l}} \\ s' \subset s_l}} e^{-E_{u_l}(s_l)} \\ &= \sum_{\substack{s' \in \Sigma^{X_v} \\ s \subset s'}} \left( e^{-\sum_{J_{i,j} \in f(v)} J_{i,j}(s'(i), s'(j))} \right) \prod_{1 \leq l \leq k} \mathcal{Z}_{u_l}(s'_{|X_{u_l} \cap X_v}) \end{aligned}$$

◀

▶ **Lemma 7.** *Let $v_1, .., v_n$ be a reversed topological ordering of $T$ starting from the root. The following algorithm samples from the Boltzmann distribution of $M$:*

*Let $s$ be the unique assignment of $\Sigma^{\emptyset}$.*
***for*** *$1 \leq i \leq n$* ***do***
  *Let $B = X_{v_i} - dom(s)$.*
  *Sample $s' \in \Sigma^B$ with probability proportional to $Z_{v_i}((s + s')_{|X_{v_i} \cap X_{pa(v_i)}})$.*
  *Extend $s$ with the values of $s'$.*
***end for***

**Proof.** To show the correctness of the algorithm, it suffices to show that at each step we have that:

$$\forall s' \in \Sigma^B : Z_{v_i}((s + s')_{|X_{v_i} \cap X_{pa(v_i)}}) \propto P(s'|s)$$

The ordering $v_1, .., v_n$ guarantees that when a node $v_i$ is visited, none of its descendants has been visited yet. Moreover, letting $u$ be the parent of $v_i$ we can split $T$ by removing $u$ from it and $v_i$ will be the root of the connected component composed of its descendants. Considering the other components together, this induces a partition of $V(T)$ as $D_{v_i} \sqcup A$ where $A$ is the set of non-descendants of $v_i$. This induces the following decomposition of the energy function:

$$E(w) = E_{v_i}(w) + E_A(v_i) \text{ with } E_A(w) = \sum_{\substack{J_{i,j} \in u \\ u \in A}} J_{i,j}(w_i, w_j)$$

that can be used to rewrite $P(s'|s)$:

$$P(s'|s) \propto \sum_{\substack{w \in \Sigma^N \\ s, s' \subset w}} e^{-E(w)} = \sum_{\substack{w \in \Sigma^N \\ s, s' \subset w}} e^{-E_{v_i}(w)} \times e^{-E_A(w)}$$

Moreover, letting $\mathcal{X}_A = \cup_{u \in A} X_u$, the decomposition of $V(T)$, together with Lemma 12 implies that: $\mathcal{X}_A \cap Y_{v_i} \subset X_v$. Except for the columns of $X_{v_i}$, $E_A$ and $E_{v_i}$ involve different sets of columns, which allows us to further simplify:

$$P(s'|s) \propto \left( \sum_{\substack{a \in \Sigma^A \\ s \subset a}} e^{-E_A(a)} \right) \times \left( \sum_{b \in \Sigma^{Y_{v_i} - X_{v_i}}} e^{-E_{v_i}(s'+s+b)} \right)$$

$$= \underbrace{\left( \sum_{\substack{a \in \Sigma^A \\ s \subset a}} e^{-E_A(a)} \right)}_{\text{constant with respect to } s'} \times \mathcal{Z}_{v_i}((s+s')_{|X_{v_i} \cap X_{pa(v_i)}}) \quad (\star)$$ ◀

▶ **Lemma 10.** *Let $v \in V(T)$ and $P_v$ be the marginal of the Boltzmann distribution restricted to the columns of $X_v \cap X_{pa(v)}$. Then for $s \in \Sigma^{X_v \cap X_{pa(v)}}$ we have that:*

$$Z_v^o(s) \times Z_v(s) = \sum_{\substack{s' \in \Sigma^{\{1,..,N\}} \\ s \subset s'}} e^{-\beta E(s')} \propto P_v(s)$$

**Proof.** If $v$ has no parent then the proof is direct: $Z_v^o(\emptyset) = 1$ and $Z_v(\emptyset) = \mathcal{Z}$. Otherwise suppose $u$ is the parent of $v$. Then consider the proof of Lemma 7 : split the decomposition $\mathcal{T}$ by removing $u$ from $T$ and assembling the non-descendants of $v$ together. Now observe that the energy function $E_A$ in the proof corresponds in fact to $E_v^o$. Then note that equation $(\star)$ directly gives the result that we seek. ◀

▶ **Lemma 11** (Augmented tree decompositions)**.** *Let $\mathcal{T}$ be a tree decomposition of $G$ of size $s \in \mathbb{N}$ and $x \in V(G)$. We can build a decomposition $\mathcal{T}^x$ of size $2s$ and treewidth at most $tw(\mathcal{T}) + 1$ such that for every $y \in V(G)$ there exists $v_y \in V(T^x)$ such that:*

$$\{x, y\} \subset X_{v_y} \cap X_{pa(v_y)}$$

**Proof.** Construct $\mathcal{T}^x$ from $\mathcal{T}$ as follows:

- Add $x$ to every bag of $\mathcal{T}$ if it is not already present.
- Add an additional child $v'$ to every node $v$ such that $X_{v'} = X_v$.

The size of bags is increased by at most 1 which justifies the bound on the treewidth of $\mathcal{T}^x$. We can verify that all the axioms of tree decompositions are verified. The first two axioms are inherited from $\mathcal{T}$, while the third is classically [13] equivalent to:

For $z \in V(G)$ the nodes of $T$ whose bag contain $z$ form a connected subtree of $T$.

Because the duplication of bags is adjacent it doesn't create further violations of this axiom, and moreover for $z = x$ the subtree is the whole tree, thus necessarily connected. It remains to be shown that $\{x, y\} \subset X_{v_y} \cap X_{pa(v_y)}$ for some $v_y \in V(T^x)$. By the first axiom of tree decompositions, there exists $v_0 \in V(T)$ such that $y \in X_{v_0}$. Take $v_1$ its correspondence in $V(T^x)$ and $v_y = v_1'$ the child of $v_1$ with the same bag. Then we have: $y \in (\{x\} \cup X_{v_0}) = X_{v_1} = X_{v_1'}$ and $x \in X_{v_1'} = X_{v_1}$ by definition. Thus $\{x, y\} \subset X_{v_y} \cap X_{pa(x_y)}$. ◀

## A.1 List of selected RNA families

Collection of 41 RFAM families considered in our case study:

| Family | N | n | PCC | $\|E\|$ | $tw$ | Family | N | n | PCC | $\|E\|$ | $tw$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| RF00001 | 230 | 712 | 0.76 | 148 | 5 | RF00003 | 203 | 100 | 0.42 | 200 | 5 |
| RF00005 | 118 | 954 | 0.83 | 129 | 6 | RF00013 | 262 | 150 | 0.39 | 200 | 3 |
| RF00019 | 145 | 104 | 0.59 | 159 | 6 | RF00020 | 192 | 180 | 0.41 | 199 | 6 |
| RF00050 | 221 | 146 | 0.37 | 200 | 3 | RF00167 | 113 | 133 | -0.059 | 128 | 6 |
| RF00169 | 127 | 261 | 0.64 | 134 | 6 | RF01051 | 136 | 155 | 0.23 | 144 | 6 |
| RF01689 | 112 | 146 | 0.3 | 133 | 6 | RF01695 | 129 | 456 | 0.68 | 135 | 6 |
| RF01699 | 264 | 194 | 0.47 | 200 | 3 | RF01731 | 193 | 210 | 0.61 | 189 | 6 |
| RF01745 | 345 | 165 | 0.38 | 200 | 1 | RF01831 | 179 | 101 | -0.026 | 193 | 6 |
| RF01852 | 119 | 109 | 0.53 | 136 | 6 | RF02034 | 248 | 172 | 0.55 | 200 | 4 |
| RF02035 | 153 | 320 | 0.78 | 152 | 6 | RF02678 | 228 | 155 | 0.52 | 200 | 3 |
| RF02921 | 200 | 143 | 0.52 | 200 | 5 | RF02928 | 130 | 236 | 0.53 | 124 | 6 |
| RF02932 | 163 | 102 | -0.03 | 169 | 6 | RF02944 | 104 | 224 | 0.68 | 123 | 6 |
| RF02957 | 232 | 448 | 0.86 | 192 | 6 | RF02960 | 240 | 155 | 0.54 | 200 | 4 |
| RF02966 | 173 | 108 | 0.16 | 179 | 6 | RF02967 | 154 | 139 | 0.37 | 159 | 6 |
| RF02968 | 114 | 229 | 0.53 | 128 | 6 | RF02972 | 128 | 154 | 0.78 | 139 | 6 |
| RF02976 | 155 | 210 | 0.79 | 156 | 6 | RF02987 | 121 | 481 | 0.72 | 129 | 6 |
| RF02996 | 118 | 167 | 0.33 | 132 | 6 | RF02999 | 102 | 248 | 0.69 | 103 | 6 |
| RF03000 | 203 | 368 | 0.5 | 200 | 5 | RF03005 | 201 | 129 | 0.39 | 170 | 6 |
| RF03021 | 119 | 141 | 0.59 | 134 | 6 | RF03074 | 117 | 261 | 0.51 | 120 | 6 |
| RF03075 | 115 | 209 | 0.72 | 130 | 6 | RF03127 | 110 | 370 | 0.59 | 125 | 6 |
| RF03147 | 112 | 149 | 0.54 | 129 | 6 | | | | | | |

$N$: alignment width, $n$: alignment height, PCC: pearson correlation between covariations of alignment and model (as in [3]), $|E|$: number of activated couplings, $tw$: final treewidth

## A.2 Processing of geometric data

The alignments provided by RFAM contain occasionally sequences annotated with PDB structures that we were able to access. These structures enable us to reconstruct distances between residues in the family. Similarly, annotation tools such as DSSR were used to reconstruct non-canonical basepairs using these tertiary structures. The distances and annotations are computed for positions relative to the real ungapped RNA sequence present in the PDB. It is then only a matter of projecting this sequence into the alignment's column space to be able to compute distances/annotations in the reference of the alignment. However, due to what we believe to be frequent annotations errors of the PDB chains, we had to perform pairwise alignments between the chains of residues actually present in the PDBs, with the sequence reported in the family alignment. Some structures were rejected when this pairwise alignment was deemed of poor quality, this explains why some families with tertiary structures available were not included. Moreover, since the structures correspond to sequences that may contain deletions, in the reference of a family's alignment, not all positions are represented in 3D structures, and some may be represented many times in different structures. When a pair of position is present in multiple structures, we choose to use the minimum distance reported, and when distances were not available for pairs of positions corresponding with a coupling, it was excluded from the analyses requiring distance information.

## A.3 Evolution of the treewidth

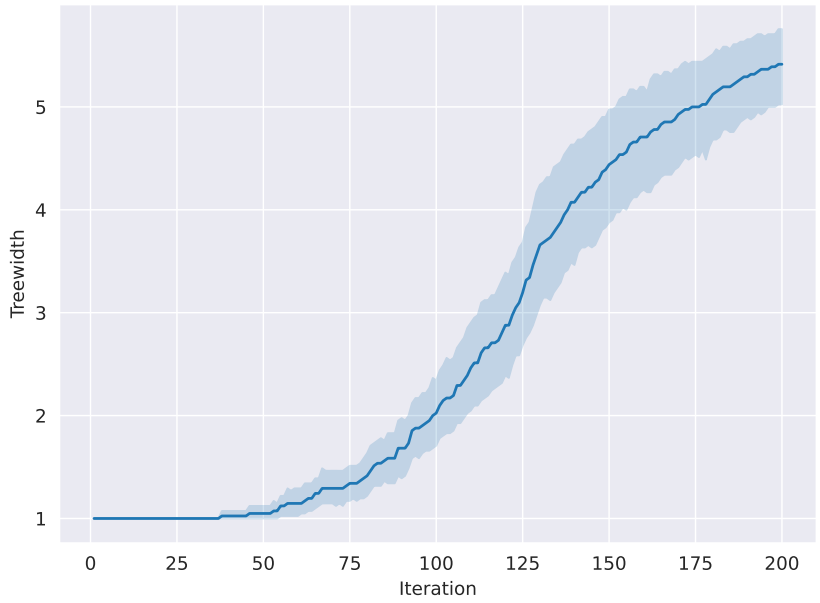



**Figure 9** Increase in average treewidth (and 95% percentiles) at different iterations. Instances reaching treewidth 6 are no longer iterated upon but are kept in the distribution.

# RNA Inverse Folding Under Stacked Base Pair Maximization

**Théo Boury**
LIX, CNRS, École polytechnique, Institut Polytechnique de Paris, Palaiseau, France

**Laurent Bulteau**
LIGM, Université Gustave Eiffel, Marne-la-vallée, France

**Yann Ponty**
LIX, CNRS, École polytechnique, Institut Polytechnique de Paris, Palaiseau, France

## Abstract

Inverse folding is a classic problem in RNA bioinformatics, crucial for designing functional synthetic RNAs, which consists in finding a sequence that uniquely folds into a target secondary structure with respect to energy minimization. In a simple base pair maximization (maxBPs) model, Bonnet *et al.* showed that a mildly constrained version of inverse folding is NP-hard. By contrast, a linear-time exact algorithm was proposed for maxBPs inverse folding, when restricted to input structures where each helix, i.e. each set of consecutive base pairs, has size at least 3. However, the maxBPs model artificially induces drastic limitations on the set of designable structures, forbidding the design of many well-known RNA families.

In this work, we adopt a more realistic energy model based on stacked base pairs and study the inverse folding under a stack maximization (maxStacks) energy model, motivated by the major contribution of stacks to RNA stability. We propose an exact $\mathcal{O}(n)$-time algorithm for maxStacks inverse folding, restricted to structures having minimum helix length $\lceil \log_{3.56}(\Delta) + 6.2 \rceil$ base pairs, where $\Delta$ is the largest degree of a loop in the target structure. Our approach hinges on the introduction of the locked property, a sufficient condition for a sequence to be a maxStacks design. Our algorithm enables the design of loops with arbitrary degree $\Delta$ in the maxStacks model, contrasting with the maxBPs model where inverse folding is unsolvable beyond $\Delta = 4$.

Interestingly, the locked property can also be utilized to partially solve maxStacks inverse folding when crossing base pairs, aka general pseudoknots, are allowed in the target structure and possible competitors. In this setting, we obtain an exact $\mathcal{O}(n)$-time algorithm for maxStacks inverse folding restricted to (pseudoknotted) targets having minimum helix length $\lceil \log_{3.56}(m) + 6.2 \rceil$, $m$ now being the number of helices. This result is surprising since checking the validity of a candidate sequence requires solving RNA folding with general pseudoknots, a problem known to be NP-hard in the maxStacks model.

We empirically evaluate the potential of maxStacks solutions by designing candidate sequences for synthetic structures, uniformly generated at random to be non-pseudoknotted for diverse minimal helix lengths. We consider a natural generalization of our exact algorithm, heuristically addressing cases where the minimum helix length condition fails, and compare it to a baseline assignment of random compatible nucleotides. Our results show that satisfying the maxStacks criterion discriminates sequences that are likely to represent solutions to the expressive Turner energy model. Moreover, sequences produced by our (generalized) algorithm are more distant, energy-wise, to their competitors than uniform compatible sequences, suggesting the potential of maxStacks designs towards complex use cases.

**2012 ACM Subject Classification** Applied computing → Molecular structural biology

**Keywords and phrases** RNA structure, RNA Design, Discrete Algorithm, String combinatorics, Stacked Base Pairs

**Digital Object Identifier** 10.4230/LIPIcs.WABI.2026.12

**Supplementary Material** *Software*: `https://doi.org/10.5281/zenodo.20266457`



© Théo Boury, Laurent Bulteau, and Yann Ponty;
licensed under Creative Commons License CC-BY 4.0





# 1 Introduction

RiboNucleic Acids (RNAs) are versatile biomolecules able both to encode genetic information and to perform catalytic functions [7]. It makes RNA a promising substrate for the fast-paced development of therapeutic tools, including mRNA vaccines, RNA interference or molecular sensors (riboswitches) [9, 16]. Mimicking such functions may require the adoption (or avoidance) of specific intricate 3D structures, motivating a rational structural design of RNAs. At its core, RNA design often involves solving inverse folding, a classic yet crucial problem in RNA Bioinformatics. RNA inverse folding consists, given a target structure $T$, in finding a sequence $\omega$ such that $T$ is the unique structure adopted by $\omega$ at its thermodynamic equilibrium, with respect to energy minimization.

An extension of inverse folding was shown to be NP-hard for secondary structures, even in the simple base pair maximization model maxBPs [2]. This difficulty motivates practical heuristics [6, 14] targeting more complex energy models [18]. In contrast, in the maxBPs model, restricting the input target $T$ to helices, i.e. sets of consecutive base pairs, of size at least 3 enables an exact $\mathcal{O}(n)$ solution for inverse folding [3]. This fuels the idea that the above-mentioned complexity results may rely on target structures being overly intricate, possibly misrepresenting the practical complexity of inverse folding in realistic settings.

Even though the maxBPs model is useful as a minimal approximation of free-energy, amenable to combinatorial studies [8, 17] and initialization strategies [4], it unfortunately falls short in terms of generalization. maxBPs postulates independent energy contributions for base pairs and, as such, disregards more complex structural motifs such as loops [18]. Loops consist of the interaction of multiple base pairs and are known to majorly contribute to structural stability. Worse, structures having at least one loop of size at least 5 or at least 3 involving unpaired positions are undesignable in the maxBPs model. Indeed, any putative solution necessarily features alternative structures, thus artificially deflating (exponentially) the set of designable structures [17]. This limitation is not without harm, as some RNAs, including tRNAs and rRNAs, that exist in nature and thus admit a sequence solution, are stated to be undesignable in maxBPs.

The above observations motivate the consideration of a more realistic energy model than maxBPs. Unfortunately, a rigorous algorithmic study of the full Turner (nearest-neighbor) model [18] remains elusive due to its large number of experimentally measured parameters (several thousands, although not always independent). As an alternative compromise, we motivate the use of the stack maximization energy model maxStacks. A stack is a loop in the Turner model, consisting of two consecutive base pairs with no unpaired position between them. Stacks are especially important in the Turner as it is the loop type which most stabilizes RNA structure, motivating its sole consideration of them preferentially to other loops types in maxStacks. Note that moving from maxBPs to maxStacks is consequential complexity-wise, since checking the validity of the solution requires solving the corresponding direct problem, RNA folding by energy minimization. Indeed, in the presence of structures with arbitrary crossings (*aka* general pseudoknots), the problem reduces to the polynomial-time solvable MAX MATCHING in maxBPs [5] but is NP-hard in maxStacks [11].

In this article, we introduce and study maxStacks inverse folding, and find that:

- The maxStacks model enables a design of structures of arbitrarily large loops, going much beyond what was feasible in maxBPs (Section 2);
- When the target and space of alternative structures are both restricted to avoid crossing interactions/pseudoknots, we find a solution for the maxStacks inverse folding in $\mathcal{O}(n)$ with $n$ size of the sequence, restricted to structures with helices having at least $\lceil \log_{3.56}(\Delta)+6.2 \rceil$ base pairs, where $\Delta$ is the largest degree of a loop in the target (Section 4);

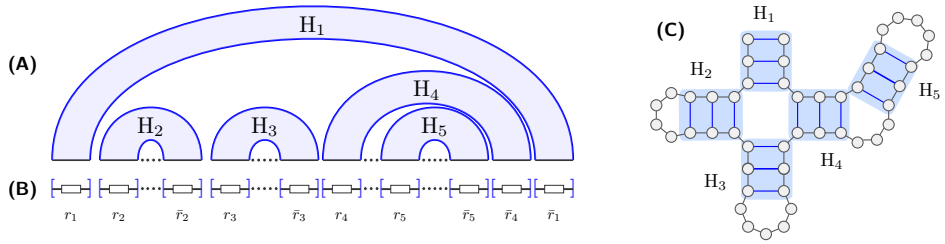



**Figure 1** (A) Secondary (2D) structure, consisting of 5 helices and maximum loop degree 4 with $\{H_1, H_2, H_3, H_4\}$. It also contains one loop of degree 2 with $\{H_4, H_5\}$ and three of degree 1 with $\{H_2\}$, $\{H_3\}$ and $\{H_5\}$; (B) Associated set of regions; (C) Classic representation of the structure.

- Empirical analyses suggest that maxStacks represents a good proxy to the Turner model for design. In particular, our newly generated maxStacks designs are more susceptible to represent direct solutions in the Turner model than random compatible sequences. This overall trend remains when the scope of our algorithm is heuristically extended to tackle helices of shorter size (Section 5);
- Considering a variant of inverse folding in maxStacks when arbitrary crossing interactions, i.e. pseudoknots, are allowed both in the target structure and alternatives, we provide a linear-time design algorithm for inverse folding in the maxStacks model for any structure having helices of length $\lceil \log_{3.56}(m) + 6.2 \rceil$, $m$ being the number of helices in the target. Notably, our algorithm guarantees the validity of the solution despite the NP-hardness of the maxStacks general folding problem (Section 6).

These results rely on the introduction of a sufficient condition (locked), introduced in Section 3, for a nucleotide sequence to be a maxStacks solution to the inverse folding problem. Overall, our results show that the stack maximization energy model maxStacks represents an interesting tradeoff between accuracy and complexity, in the context of RNA inverse folding, enabling an exact design of restricted but relevant sets of structures.

## 2 Problems statement, definitions and motivation

As stated above, an RNA *sequence* is a string $\omega \in \Sigma^n, \Sigma := \{\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{U}\}$ of length $n$. An RNA *structure* $T$ consists of a set of base pairs, *i.e.* pairs of integers $(i, j) \in [1, n]^2$, $j > i + 1$ that are pairwise disjoint. A structure $T$ is *compatible* with a sequence $\omega$ iff $\forall (i, j) \in T : \{\omega[i], \omega[j]\} \in \mathcal{B}$ with $\mathcal{B} := \{\{\mathsf{A}, \mathsf{U}\}, \{\mathsf{G}, \mathsf{C}\}, \{\mathsf{G}, \mathsf{U}\}\}$. A *crossing* in a structure consists of two base pairs $(i, j), (k, l) \in T$ such that $i < k < j < l$. A structure is *secondary* (or *2D*) if it does not have any crossing. Without this restriction, a structure is said to be *pseudoknotted* (abbreviated *PK*).

As illustrated in Figure 1, a structure $T$ (2D or not) can be uniquely decomposed into a set $\mathcal{H}_T$ of *helices*, each helix $H = \{(i, j), (i + 1, j - 1), \ldots, (i + h - 1, j - h + 1)\} \subset T$ being a maximal subset of consecutive base pairs, with $h = |H|$ denoting the size of helix $H$.

A *stacked base pair*, or *stack* in short, consists of two consecutive base pairs $(i, j), (i+1, j-1)$ occurring in a structure $T$, and we denote by $\mathcal{S}_T$ the set of stacks induced by $T$. Note that a helix of size $h$ consists of $h - 1$ stacks, so we have $|\mathcal{S}_T| = \sum_{H \in \mathcal{H}_T} |H| - |\mathcal{H}_T|$. A structure is *admissible* if each base pair belongs to at least one stack, i.e. $2 \leq h \leq \frac{n}{2}$.

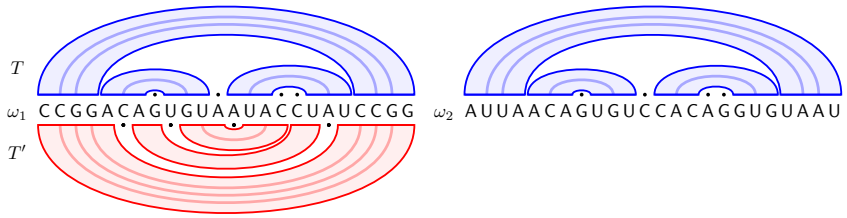



**Figure 2** A target structure $T$ (top) with 3 helices for a total of 7 stacks, and two sequences $\omega_1$, $\omega_2$ compatible with $T$. Sequence $\omega_1$ (left) is not a design since there exists an alternative structure $T'$ with the same number of stacks. Sequence $\omega_2$ (right) is a solution for 2D-Inverse Folding($T$).

▶ **Problem 1** ((maxStacks) 2D-Inverse Folding).
**Input:** *Admissible target 2D structure $T$; length $n$.*
**Output:** *RNA sequence $\omega \in \Sigma^n$, compatible with $T$, such that if $T' \neq T$ is an admissible 2D structure compatible with $\omega$ then $|\mathcal{S}_T| > |\mathcal{S}_{T'}|$; or $\varnothing$ if no such sequence exists.*

By abuse of language, we sometimes say that a sequence $\omega$ is a *design* to say that $\omega$ is a solution to 2D-Inverse Folding.

Although not formally proven here, we strongly suspect the problem 2D-Inverse Folding to be NP-hard, at least in a mildly generalized version including position-specific constraints. Indeed, Bonnet *et al* [2] showed the NP-hardness of a similarly extended version of maxBPs inverse folding, where sequences are designed to avoid alternatives structures with equal or more base pairs than the target.

We primarily focus on 2D-Inverse Folding in this manuscript so any structure is intended as 2D unless mentioned otherwise. However, we do study the variant where pseudoknots are allowed both in the target and in the alternatives. In Section 6, we give the necessary formalism and prove that our approach extends to this pseudoknotted setting with minimal modifications.

**Further definitions: match, regions and loops.** For any interval $I \subseteq [1, n]$, we write $I_{\text{start}}$ and $I_{\text{end}}$ respectively for its first and last index (i.e. $I = [I_{\text{start}}, I_{\text{end}}]$). The *complement* of $x = \mathsf{A}$ (resp. $\mathsf{U}$, $\mathsf{C}$, $\mathsf{G}$) is $\overline{x} = \mathsf{U}$ (resp. $\mathsf{A}$, $\mathsf{G}$, $\mathsf{C}$). The *reverse complement* of string $S = x_1 \dots x_\ell$ is $\text{RevComp}(S) = \overline{x_\ell} \dots \overline{x_1}$. For a helix $H = \{(i, j), \dots (i + h - 1, j - h + 1)\}$, intervals $r = [i, i + h - 1]$ and $r' = [j - h + 1, j]$ are the *regions* of $H$, and we write $r' = \overline{r}$ and $r = \overline{r'}$. We write $\mathcal{R}_T$, or simply $\mathcal{R}$ whenever clear, for the overall set of (pairwise disjoint) regions occurring in $T$. Since a helix is fully determined by its two regions, we slightly abuse notations and write $H = (r, \overline{r})$ for the helix with regions $r$ and $\overline{r}$ (in the absence of further precision, we choose $r$ as the leftmost interval among both regions of $H$). See Figure 1 for an illustration of those definitions.

Given a target $T$ and a sequence $\omega$, we say that an interval $I = [a, a + h]$ *matches* (resp. *aligns with*) interval $[b-h, b]$ in $T$ if for all $k \in [0, h-1]$, $\{\omega[a+k], \omega[b-k]\} \in \{\{\mathsf{A}, \mathsf{U}\}, \{\mathsf{G}, \mathsf{C}\}\}$ (resp. $\in \mathcal{B}$). Finally, we use $\omega[r] := (\omega[a], \cdots, \omega[b])$ to denote the substring associated with a region $r = [a, b]$. With such notations, the compatibility criterion becomes: for each $r \in \mathcal{R}$, $r$ aligns with $\overline{r}$.

Let $r, r' \in \mathcal{R}_T$. They are *adjacent* if $r'_{\text{start}} = r_{\text{end}} + 1$. Also, $r'$ is the *successor* of $r$ (and $r$ the *predecessor* of $r'$) if $r'$ is the first region to the right of $\overline{r}$. This relation on the set of regions $\mathcal{R}_T$ yields a graph with connected components. Since each region can have at most

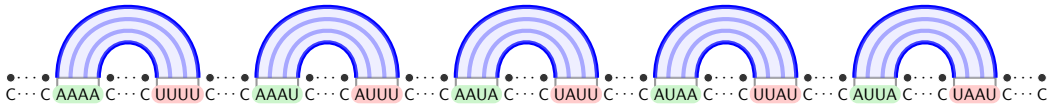



**Figure 3** Loops of arbitrary degree $\Delta$ can be designed in the maxStacks model using helices of size at least $1 + \lceil \log \Delta \rceil$. For instance, a solution sequence exists for a target with loop of degree $\Delta = 5$, and helix size $h = 4$, while such a structure cannot be designed in the maxBPs model.

one successor and predecessor, the components are only paths and cycles. All helices with a region in the same component are called a *loop*. Note that each helix $(r, \overline{r})$ belongs to at most two loops: one for the connected component containing $r$, one for the one containing $\overline{r}$. The size (number of helices) of a loop is called its *degree*. A loop is further assumed to *include* any unpaired position (i.e. not in any region) between two of its successive regions (see Figure 1 for an example of loops). The 2D setting further implies the following useful property: given a helix $(r, \overline{r})$, we can define an inside $([r_{\text{start}}, \overline{r}_{\text{end}}])$ and an outside $([1, r_{\text{end}}] \cup [\overline{r}_{\text{start}}, n])$, overlapping only on the helix itself, and all helices and loops are included in either side by the absence of crossings.

**The maxStacks model enables a design of arbitrary degree loops for large helices.** Previous work [8] showed that the maxBPs model fails to produce designs for any target structure with either: i) a loop of degree at least 5; or ii) a loop of degree at least 3, including at least 1 unpaired position [8]. This limitation is largely detached from reality, as such motifs can be found in virtually any RNA structure, including tRNAs, 5s ribosomal RNA…

Contrasting, loops of arbitrarily large degrees can be designed for maxStacks using sufficiently long helices, as illustrated by Figure 3. Such an observation generalizes to any secondary structure, possibly including pseudoknots/crossing interactions as long as sufficiently long helices are considered (see Theorem 5).

However, all helices in the same loop need to be designed with distinct sequences, leading to the following logarithmic lower-bound on the helix size.

▶ **Lemma 1.** *Let $T$ be a 2D structure containing a degree-$\Delta$ loop with helices of size $h$ with $h < \log_6(\Delta)$. Then $T$ is not designable in maxStacks.*

Intuitively, due to base pairing constraints, there is only $6^h$ possible assignments for a helix of size $h$. Thus, $h < \log_6(\Delta)$ implies a collision, *i.e.* identical assignments to two helices, leading to an alternative structure with equal (or greater) number of stacks than the target. We postpone to Appendix C a formal proof of this result.

## 3 Sufficient conditions for Inverse Folding in the maxStacks model

In this section we present the *locked* criterion for sequences and prove that satisfying this criterion is sufficient to yield a design. This criterion is based on the presence of a *key* within each region, with strong constraints on how keys can be matched within the rest of the sequence, thus forbidding "global" rearrangements where whole regions are entirely mapped differently. The non-extension condition, on the other hand, is crucial to forbid "local" rearrangements that would locally increase an existing helix by a few stacks. We first give a general criterion that applies to any target structure and prove its core property in Lemma 2 (this Lemma will further be used in Section 6 in the pseudoknotted setting). We then specify the criterion into *2D-locked*, leading finally to Theorem 3 for 2D structures.

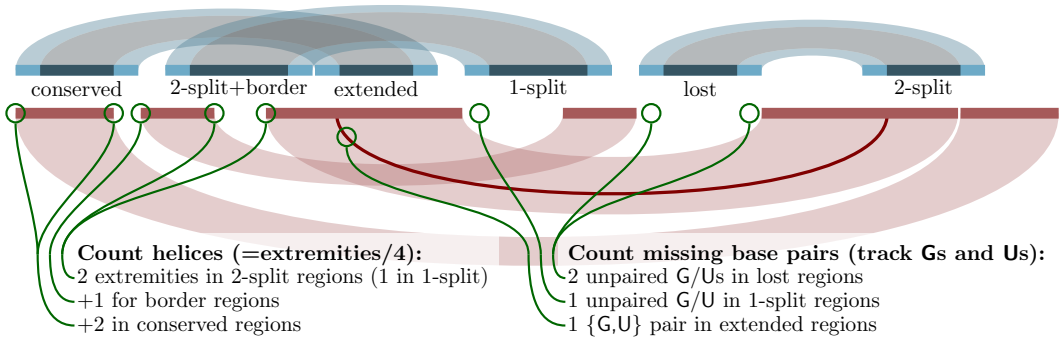



**Figure 4** Intuition behind the proof of Lemma 2. Top: a locked sequence for the target structure $T$ shown with one interval per region, and a darker stripe for the key. Bottom: alternative structure $T'$. To show that any such $T'$ has fewer stacks than $T$, we carefully count helices and base pairs in $T'$, with a case distinction for each region of $T$ depending on its overlap with regions of $T'$.

▶ **Definition 1** (locked). *Given a sequence $\omega \in \Sigma^n$ and a structure $T$ (2D or not), we say that $\omega$ satisfies the locked property (with respect to $T$) if, for each $r \in \mathcal{R}$, there exists an interval $\lambda^r \subset r$, named the* key *of $r$, such that:*

1. ***Base compatibility:*** *$\omega$ is compatible with $T$, using only $\{A, U\}$ and $\{G, C\}$ base pairs;*
2. ***Unpaired A or C:*** *All unpaired positions are assigned A or C;*
3. ***Helix keys:*** *$\forall\ r \in \mathcal{R}$, each interval $[r_{\text{start}}, \lambda^r_{\text{start}}]$ and $[\lambda^r_{\text{end}}, r_{\text{end}}]$ contains at least one position $i$ with $\omega[i] \in \{G, U\}$;*
4. ***Keys match keys:*** *$\forall\ r \in \mathcal{R}$, if $\lambda^r$ matches any interval $I \subset [1, n]$, then $\exists r' \in \mathcal{R}$, $I = \lambda^{r'}$;*
5. ***No Interval Extension:*** *$\forall r \in \mathcal{R}$, if an interval $I$, $\lambda^r \subseteq I$ and $I \not\subseteq r$, aligns with any other interval $I'$, then the alignment between $I$ and $I'$ uses at least one pair $\{G, U\}$.*

This last condition can be informally rephrased as: if any alternative helix properly overlaps (and extends) a target region $r$, then it cannot contain the whole key $\lambda^r$.

▶ **Lemma 2** (Key Matching Lemma). *Let $T$ be a structure and $\omega$ be a locked sequence for $T$. For any $T'$ compatible with $\omega$, if $|\mathcal{S}_T| \leq |\mathcal{S}_{T'}|$, then all positions labeled G or U are paired in $T'$ (without any $\{G, U\}$ pair), and each key $\lambda^r$ of $\omega$ matches $\lambda^{r'}$ in $T'$ with some $r' \in \mathcal{R}_T$.*

**Proof.** The overall proof strategy is depicted in Figure 4. We write $\rho = |\mathcal{R}_T|$ for the number of regions of $T$, $\pi_{\mathsf{GU}}$ for the number of positions belonging to a pair $\{\mathsf{G}, \mathsf{U}\}$ in $T$, and $\mu$ (resp. $\mu_{\mathsf{GU}}$, $\mu_{\mathsf{AC}}$) for the number of unpaired positions in $T$ (resp. unpaired positions with label in $\{\mathsf{G}, \mathsf{U}\}$ or $\{\mathsf{A}, \mathsf{C}\}$ in $\omega$), i.e. $\mu = \mu_{\mathsf{GU}} + \mu_{\mathsf{AC}}$. We use the corresponding primed notations for $T'$ ($\rho'$, $\pi'_{\mathsf{GU}}$, $\mu'$, $\mu'_{\mathsf{GU}}$ and $\mu'_{\mathsf{AC}}$). We first show the following:

$$|\mathcal{S}_T| - |\mathcal{S}_{T'}| = \frac{\mu' - \mu + \rho' - \rho}{2} \tag{1}$$

Indeed, there are $n - \mu$ paired positions in $T$, thus accounting for $\frac{n-\mu}{2}$ base pairs. Each size-$h$ helix contributes to $h - 1$ stacks, so the overall number of stacks is the number of base pairs minus the number of helices. There are $\frac{\rho}{2}$ helices in $T$, so $|\mathcal{S}_T| = \frac{n-\mu-\rho}{2}$ and similarly $|\mathcal{S}_{T'}| = \frac{n-\mu'-\rho'}{2}$, thus yielding Equation (1).

We first compare numbers of unpaired positions with label A or C in $T$ and $T'$. Bases A or C can only be paired with G and U. Writing $n_{\mathsf{GU}}$ for the total number of Gs and Us in $\omega$, in $T$, there are $n_{\mathsf{GU}} - \pi_{\mathsf{GU}} - \mu_{\mathsf{GU}}$ pairs in $\{\{\mathsf{A}, \mathsf{U}\}, \{\mathsf{C}, \mathsf{G}\}\}$ (resp. $n_{\mathsf{GU}} - \pi'_{\mathsf{GU}} - \mu'_{\mathsf{GU}}$ pairs in $T'$). Thus,

the difference in number of unpaired A and C is: $\mu'_{\mathsf{AC}} - \mu_{\mathsf{AC}} = (\pi'_{\mathsf{GU}} + \mu'_{\mathsf{GU}}) - (\pi_{\mathsf{GU}} + \mu_{\mathsf{GU}})$. By the locked property, $\pi_{\mathsf{GU}} = 0$ (Condition 1), $\mu_{\mathsf{GU}} = 0$ (Condition 2), so $\mu'_{\mathsf{AC}} - \mu_{\mathsf{AC}} = \pi'_{\mathsf{GU}} + \mu'_{\mathsf{GU}}$. Overall, Equation (1) can then be improved into the following:

$$|\mathcal{S}_T| - |\mathcal{S}_{T'}| = \frac{\mu'_{\mathsf{AC}} + \mu'_{\mathsf{GU}} - \mu_{\mathsf{AC}} - \mu_{\mathsf{GU}} + \rho' - \rho}{2} = \frac{\pi'_{\mathsf{GU}} + 2\mu'_{\mathsf{GU}} + \rho' - \rho}{2} \tag{2}$$

We then focus on regions of $T$ and $T'$. We partition regions of $T$ into the following categories, so that $\rho = \rho_{\simeq} + \rho_{\subset} + \rho_2 + \rho_1 + \rho_0$:

- $\rho_{\simeq} \to$ *conserved* regions $r \in \mathcal{R}_T$ for which there exists $r' \in \mathcal{R}_{T'}$ with $\lambda^r \subseteq r' \subseteq r$
- $\rho_{\subset} \to$ *extended* regions $r \in \mathcal{R}_T$ for which there exists $r' \in \mathcal{R}_{T'}$ with $\lambda^r \subset r'$ and $r' \not\subseteq r$
- $\rho_2 \to$ *2-split* regions $r \in \mathcal{R}_T$ that are neither conserved nor extended, and that intersect at least two distinct regions in $\mathcal{R}_{T'}$.
- $\rho_1 \to$ *1-split* regions $r \in \mathcal{R}_T$ that are neither conserved nor extended, and that intersect exactly one region in $\mathcal{R}_{T'}$
- $\rho_0 \to$ *lost* regions $r \in \mathcal{R}_T$ that do not intersect any region in $\mathcal{R}_{T'}$

Further, a 2-split or extended region is a *border* if it is the first region in $T$ or if the region directly to its left is conserved, we write $\beta$ for the number of border regions. There are overall $2\rho'$ region extremities in $T'$, with, by definition, at least 2 extremities per conserved and 2-split regions of $T$, and at least one extremity per 1-split region. Plus, for a border region $r$, the left extremity of the left-most region of $T'$ intersecting $r$ is not counted in the above (it is either unpaired in $T$ or in a conserved or extended region by definition of border, and it would count as an additional extremity in such a region). Overall, it leads to the following equation:

$$2\rho' \geq 2\rho_{\simeq} + 2\rho_2 + \rho_1 + \beta. \tag{3}$$

Consider now a 1-split or lost region $r$. By Condition 3, both intervals $[r_{\text{start}}, \lambda^r_{\text{start}}]$ and $[\lambda^r_{\text{end}}, r_{\text{end}}]$ contain a position $i$ with $\omega[i] \in \{\mathsf{G}, \mathsf{U}\}$. If $r$ is lost, then it contains at least two positions labeled $\{\mathsf{G}, \mathsf{U}\}$ unpaired in $T'$. If $r$ is 1-split and intersecting $r' \in \mathcal{R}_{T'}$, then $r'$ cannot intersect both intervals (otherwise it would also contain $\lambda^r$ and $r$ would be conserved or extended), so $r$ features at least one $\{\mathsf{G}, \mathsf{U}\}$ unpaired in $T'$. Thus

$$\mu'_{\mathsf{GU}} \geq 2\rho_0 + \rho_1. \tag{4}$$

Finally by Condition 5, if $r$ is extended with $r'$, then it features at least one endpoint of a $\{\mathsf{G}, \mathsf{U}\}$-pair in $T'$, i.e.

$$\pi'_{\mathsf{GU}} \geq \rho_{\subset}. \tag{5}$$

Combining the inequalities (3), (4) and (5) with Equation (2), we get:

$$\begin{aligned}
|\mathcal{S}_T| - |\mathcal{S}_{T'}| &= \frac{1}{2}(\pi'_{\mathsf{GU}} + 2\mu'_{\mathsf{GU}} + \rho' - \rho) \\
&\geq \frac{1}{2}(\pi'_{\mathsf{GU}} + 2\mu'_{\mathsf{GU}} + (\rho_{\simeq} + \rho_2 + \frac{\rho_1}{2} + \frac{\beta}{2}) - (\rho_{\subset} + \rho_{\simeq} + \rho_2 + \rho_1 + \rho_0)) \\
&\geq \frac{1}{2}(\pi'_{\mathsf{GU}} + 2\mu'_{\mathsf{GU}} + \frac{\beta}{2} - (\rho_{\subset} + \frac{1}{2}\rho_1 + \rho_0)) \\
&\geq \frac{1}{2}(\pi'_{\mathsf{GU}} + 2\mu'_{\mathsf{GU}} + \frac{\beta}{2} - (\pi'_{\mathsf{GU}} + \frac{1}{2}\mu'_{\mathsf{GU}})) \\
&\geq \frac{3}{4}\mu'_{\mathsf{GU}} + \frac{\beta}{2}
\end{aligned} \tag{6}$$

We now use $|\mathcal{S}_{T'}| \geq |\mathcal{S}_T|$: so $\mu'_{\mathsf{GU}} = 0$ and $\beta = 0$. This implies $\rho_1 = \rho_0 = 0$ with inequality (4), so all regions of $T$ are either conserved, 2-split or extended. Moreover, $\beta = 0$: there can be no 2-split or extended region (otherwise the leftmost 2-split or extended region would be a border), and all regions of $T$ are conserved. Thus each $\lambda^r$ is included in some region of $T'$ and each region of $T'$ contains at most one $\lambda^r$ (by the absence of extensions), yielding $\rho' \geq \rho$. Using Equation (2), we obtain $\pi'_{\mathsf{GU}} = 0$. So for any $r \in \mathcal{R}_T$, $T'$ matches (without $\{\mathsf{G},\mathsf{U}\}$ pairs) $\lambda^r$ with an interval $I$, and by Condition 4, $I = \lambda^{r''}$ for some $r'' \in \mathcal{R}_T$. ◀

▶ **Definition 2.** *A sequence $\omega$ is* 2D-locked *for structure $T$ if $\omega$ is locked, with the additional condition 4a below that overrides original condition 4:*

**4a** ***Loop Unique Keys:*** *$\forall r \in \mathcal{R}$, $\forall \lambda' \neq \lambda^{\overline{r}}$ such that $\lambda^r$ matches $\lambda'$ then $\exists r' \in \mathcal{R}, \lambda' = \lambda^{r'}$ and $r, r'$ are not in the same loop.*

▶ **Theorem 3.** *Given a 2D structure $T$, if a sequence $\omega$ is 2D-locked, then $\omega$ is a solution of* 2D-Inverse Folding *$(T)$.*

**Proof.** We first show the following claim:

▷ Claim 4. For any alternative $T'$ with $|\mathcal{S}_{T'}| \geq |\mathcal{S}_T|$ and helix $(r, \overline{r})$, $T'$ matches $\lambda^r$ with $\lambda^{\overline{r}}$.

Proof. We consider a *wrong match* as a pair of keys $\lambda^r$, $\lambda^{r'}$ with $r$ and $r'$ from distinct helices (necessarily from distinct loops with condition 4a), with $\lambda^r$ and $\lambda^{r'}$ matched together in $T'$. We consider a *deepest* wrong match, such that no other wrong match occurs in the inside of the helix formed with this match in $T'$. Due to $\lambda^r$, $\lambda^{r'}$ being from different loops in $T$, the interval $I = [\lambda^r_{\text{start}}, \lambda^{r'}_{\text{end}}]$ contains another key $\lambda^x$, whose match $\lambda^{\overline{x}}$ (in $T$) is not in $I$. By Lemma 2, $\lambda^x$ matches perfectly another key in $I$, thus different from $\lambda^{\overline{x}}$, and $\lambda^x$ also forms a wrong match: a contradiction with $\lambda^r$, $\lambda^{r'}$ chosen as the deepest wrong match in $T'$, which concludes the proof. ◁

It remains to show that the region endpoints in $T'$ are identical to those in $T'$. By condition 5, the region of $T'$ containing $\lambda^r$ is necessarily included in $r$. If it is strictly included, then at least two bases paired in $T$ become unpaired in $T'$ so $T'$ contains at least one unpaired $\mathsf{GU}$, which is excluded by Lemma 2. Then the region of $T'$ containing $\lambda^r$ is equal to $r$, and $T'$ matches the same regions as $T$, so $T' = T$. ◀

## 4 Producing locked sequences for targets with sufficiently large helices

With the 2D-locked criterion in hand, we can now build an algorithm, see Algorithm 1, producing such sequences, and thus designs. The main result is summarized below.

▶ **Theorem 5.** *If $T$ is a 2D structure over $n$ nucleotides featuring helices of size above $\lceil \log_{3.56}(m) + 6.2 \rceil$, with $m$ maximum degree of a loop in $T$, then Algorithm 1 produces a solution to* 2D-Design *in $O(n)$ time and space.*

**Proof.** Algorithm 1 builds a 2D-locked sequence in linear time and space by Lemmas 8 and 9, provided input helices are large enough, and such output is a solution by Theorem 3. ◀

▶ Remark 6. Our maxStacks algorithm unlocks the design of structures with large multiloops that, to the knowledge of the author, were not possible to design exactly to this day. Indeed, one can design any structure featuring multiloops of size 9, as long as helices are of size 8, and virtually any multiloop (at least the ones known to this day) when helices are of size 9 or more. It positively contrasts with maxBPs, where any structure with multiloops of size 5 or more is undesignable, independently from any restriction on helix sizes. (see Table 1 in Appendix)

**Algorithm 1** Algorithm to get sequences that are designs in maxStacks.

**Input:** RNA structure $T$ with "large enough" helices, sequence length $n$
**Output:** Design $\omega$ for $T$ that is 2D-locked

**function** DESIGN($T$, $n$)
  $\omega = X^n$
  $\omega[i] = \mathsf{C}$ for all unpaired position $i$ in $T$
  Pick nodes $P_r$ in $\mathcal{N}$ (Figure 5) for each $r \in \mathcal{R}_T$ such that :
  – if $\overline{r}_{\text{end}} + 1 = r'_{\text{start}}$ for some $r' \in \mathcal{R}_T$, then $\mathcal{N}$ has an arc $(P_r, P_{r'})$,
  – if position $r_{\text{start}} - 1$ is unpaired or $< 1$, then $P_r$ is an enter-node
  – if position $\overline{r}_{\text{end}} + 1$ is unpaired or $> n$, then $P_r$ is an exit-node
  – $P_r \neq \mathsf{CU}$ or $P_{\overline{r}} \neq \mathsf{CU}$
  Pick a *key core* $k_H \in \{\mathsf{A}, \mathsf{C}, \mathsf{U}, \mathsf{G}\}^*$ for each helix $H$ in $T$ such that:
  – all strings $k_H$ have the same length
  – no string $k_H$ starts with $\mathsf{A}$ or contains $\mathsf{AA}$ or $\mathsf{UU}$
  – $k_H \neq k_{H'}$ for any two helices $H$, $H'$ in the same loop.
  **for** each helix $H = (r, \overline{r})$ of $T$ **do**
    **if** $P_{\overline{r}} = \mathsf{CU}$ **then**
      $(r, \overline{r}) = (\overline{r}, r)$
    $\omega[r] = P_r + \mathsf{AAA}^* + k_H + \text{RevComp}(P_{\overline{r}})$;
      ▷ $\mathsf{AAA}^*$ = fill remaining positions with $\mathsf{A}$s, minimum 2
    $\omega[\overline{r}] = \text{RevComp}(\omega[r])$.
  **return** $\omega$

The overall strategy is as follows (see Figure 6 for an example): each key $\lambda^r$ contains a string composed of $\mathsf{AA}$ followed by a unique string unique in each local loop. Using distinct keys results in the logarithmic lower bound on the helix size given in Table 1 ($k \geq \log_{3.56}(m) + 6.2$), within a constant factor of the theoretical lower bound provided by Lemma 1. Pattern $\mathsf{AA}$ (and its reverse complement $\mathsf{UU}$) only occurs at such positions, so keys can only match keys, as required by conditions 4b and 4a. The design of region borders is controlled by the graph $\mathcal{N}$, defined below: each region is given a prefix corresponding to a node in $\mathcal{N}$, and transitions between consecutive regions follow transitions in $\mathcal{N}$. This way, any interval extension beyond the bounds of the region would imply a certain pattern within the graph, and any such pattern has been carefully avoided in the construction of $\mathcal{N}$, as formalized in Observation 7 below.

▶ **Observation 7.** *Let $\mathcal{N}$ be the graph defined in Figure 5. Nodes $GC, CG$ are* enter-nodes *and $GC, UC, CG$ are* exit-nodes*. Nodes $\{UC\}, \{AG\}, \{CG, CU\}, \{GC\}$ are respectively called U-, A-, C- and G-nodes. Then $\mathcal{N}$ satisfies the following properties (with the intuitive purpose for each property indicated above).*

▷ Force a base in $\{\mathsf{G}, \mathsf{U}\}$ in each node and each node complement:

1. *Each node has one base in $\{G, U\}$ and the other in $\{A, C\}$*

▷ Avoid forbidden patterns $\mathsf{AA}$ and $\mathsf{UU}$:

2. *No node ends with $A$ and only $CU$ ends with $U$.*
3. *There is no arc between $A$- and $U$-nodes in either direction*

▷ Avoid helix extensions with a pair $(x, \overline{x})$, $(\mathsf{G}, \mathsf{U})$ or $(\mathsf{U}, \mathsf{G})$ from adjacent regions:

4. *No node has both in- and out-arcs to $x$-nodes for any $x \in \{A, U, C, G\}$.*
5. *No node has an in-arc from a $C$-node and an out-arc to a $U$-node.*
6. *No node has an in-arc from an $A$-node and an out-arc to a $G$-node.*

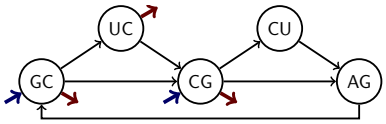



**Figure 5** Graph $\mathcal{N}$ of adjacent region extremities (cf Observation 7). Algorithm 1 assigns the nodes to all region prefixes, the suffix of matching regions being known by complementarity. Enter-nodes (resp. exit-nodes) are marked with incoming (outgoing) arrows. There exist paths from enter- to exit-nodes and cycles of any lengths (all avoiding CU except size-5 cycles).

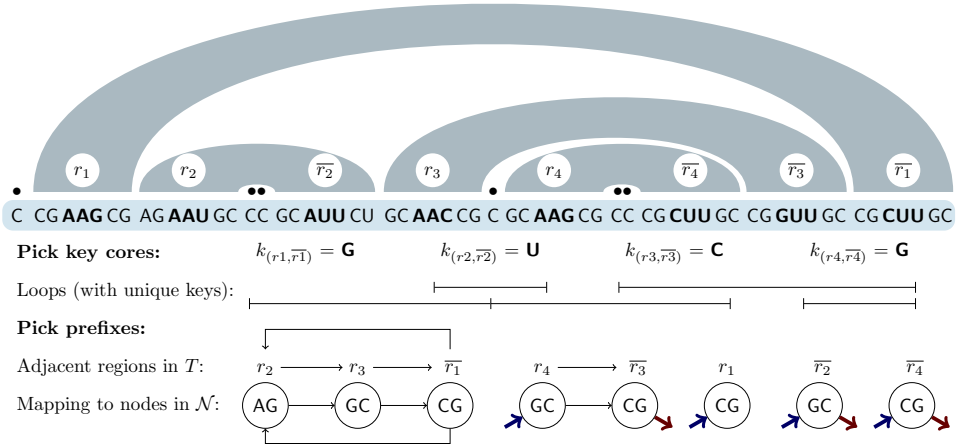



**Figure 6** Execution of Algorithm 1 on a sample target 2D structure with 4 size-7 helices (top). Helices are shown as wide arcs and unpaired positions as bullets. The resulting design is aligned with the target, with prefixes $P_r$ and suffixes RevComp($P_{\overline{r}}$) in light font, and the rest (including key cores/reverse complements) in bold. In the result, the key cores are unique within each loop, and prefixes are picked in graph $\mathcal{N}$ according to region adjacencies.

▷ Avoid helix extensions with a pair (C, G) using an unpaired C:

7. *No enter-node has an out-arc to a G-node*
8. *No exit-node has an in-arc from a C-node*

▷ Assign nodes to any path or cycle, avoiding CU whenever possible:

9. *For any $k \geq 1$, there exists a size-$k$ path avoiding CU from an enter- to an exit-node.*
10. *For any $k \geq 3$, there exists a size-$k$ cycle containing at most one occurrence of CU.*

We show the algorithm correctness in two steps: first in Lemma 8 we show that the algorithm does produce some output if helices are large enough (proof in Appendix D). Then in Lemma 9 we show that the output is indeed a locked sequence (PK-locked in the PK case and 2D-locked in the 2D case).

▶ **Lemma 8.** *If each helix in $T$ has size at least $\lceil \log_{3.56}(m) + 6.2 \rceil$, with $m$ being the maximum loop degree, then Algorithm 1 does produce some output string $\omega$ in linear time.*

▶ **Lemma 9.** *Output $\omega$ is 2D-locked.*

**Proof.** In the following we use the notation $H = (r, \overline{r})$ where $r$ is chosen as in the algorithm, i.e. so that $P_{\overline{r}} \neq$ CU. We write $\ell$ for the length of all key cores. With lines 16 and 18 of the algorithm, we have:

$$\omega[r] = P_r\ \mathsf{A}^i\ k_H\ \text{RevComp}(P_{\overline{r}}); \qquad \omega[\overline{r}] = P_{\overline{r}}\ \text{RevComp}(k_H)\ \mathsf{U}^i\ \text{RevComp}(P_r). \tag{7}$$

where $i \geq 2$, $P_{\overline{r}} \neq$ CU, strings $P_r$ and $k_r$ satisfy respectively the conditions of lines 4-12.

▷ Claim 10. There are no occurrences of AA or UU in $\omega$, other than those included in $\mathsf{A}^i$ and $\mathsf{U}^i$ for each helix.

▷ Claim 11. If regions $r$ and $r'$ satisfy $P_r = P_{r'}$, then $\{\omega[r_{\text{start}} - 1], \omega[\overline{r'}_{\text{end}} + 1]\}$ is not a valid base pair (or some position is out of interval $[1, n]$)

These both claims are consequences of Observation 7 and are detailed in Appendix D.

We now prove that the successive conditions are met for the sequence $\omega$ to be locked. Condition 1 is clear by Line 18. For Condition 2, Line 3 sets C to all unpaired positions, and lines 16 and 18 only affect paired positions.

**Condition 3.** Each region starts and ends with a node in $\mathcal{N}$ (or its reverse complement), and by Observation 7.1 each node contains a base in $\{\mathsf{G}, \mathsf{U}\}$ and another in $\{\mathsf{A}, \mathsf{C}\}$ so its reverse complement also contains a base in $\{\mathsf{G}, \mathsf{U}\}$. We thus define, for each region $r \in \mathcal{R}$, $\lambda^r = [r_{\text{start}} + 1, r_{\text{end}} - 1]$. Then, $\omega[\lambda^r] = x\mathsf{A}^i k y$ or $\omega[\lambda^r] = \overline{y}\text{RevComp}k\mathsf{U}^i\overline{x}$ for some base $x \neq \mathsf{A}$, some $i \geq 2$ and some key core $k$.

**Condition 4a.** Let $H = (r, \overline{r})$. Assume that $\lambda^r$ can be matched to an interval $I$ (the proof is similar for $\lambda^{\overline{r}}$). Then $\omega[I] = \text{RevComp}(x\mathsf{A}^i k_H y) = \overline{y}\text{RevComp}(k_H)\mathsf{U}^i\overline{x}$ with $i \geq 2$ and $x \neq \mathsf{A}$ so $\overline{x} \neq \mathsf{U}$. By Claim 10, substring UU necessarily appears in $\omega$ within an interval $\mathsf{U}^j$ for some key in helix $H' = (r', \overline{r'})$, and since $\overline{k_H}$ does not end with U (condition line 11) and $\overline{x} \neq \mathsf{U}$, we necessarily have $i = j$ and $I = \lambda^{\overline{r'}}$. Moreover, $\text{RevComp}(k_{H'}) = \text{RevComp}(k_H)$, so $k_H = k_{H'}$. Line 12 implies that $H$ and $H'$ are in distinct loops, thus satisfying Condition 4a.

**Condition 5.** Let $(r, \overline{r})$ be a helix. Assume that an interval $I \supseteq \lambda^r$ aligns with an interval $I'$, without using any $\{\mathsf{G}, \mathsf{U}\}$ pair within region $r$, which means that $I \cap r$ matches the corresponding sub-interval of $I'$. We show that $I \subseteq r$. By Condition 4, $I'$ contains an interval $\lambda^{\overline{r'}}$ for some helix $(r', \overline{r'})$ with $\lambda^r$ matching with $\lambda^{\overline{r'}}$. Thus, both keys have the same size and with $\forall l, P_l \in \mathcal{N}$ of same size, we have $|[r'_{\text{start}}, \lambda^{r'}_{\text{end}}]| = |[\lambda^r_{\text{start}}, r_{\text{end}}]|$. We first show that $I_{\text{end}} \leq r_{\text{end}}$: indeed, this is clear if $I_{\text{end}} < r_{\text{end}}$. Otherwise, we have the following relations:

$$\omega[\overline{r'}_{\text{start}}, \overline{r'}_{\text{start}} + 1] = \text{RevComp}(\omega[r_{\text{end}} - 1, r_{\text{end}}]) \quad \text{(match with } I \text{ within region } r\text{)}$$
$$\omega[\overline{r'}_{\text{start}}, \overline{r'}_{\text{start}} + 1] = \text{RevComp}(\omega[r'_{\text{end}} - 1, r'_{\text{end}}]) \quad \text{(by construction of } \omega[r'], \omega[\overline{r'}]\text{)}$$

so $\omega[r_{\text{end}} - 1, r_{\text{end}}] = \omega[r'_{\text{end}} - 1, \overline{r'}_{\text{end}}]$ and $P_{\overline{r}} = P_{r'}$. Thus by Claim 11, $\{\omega[\overline{r'}_{\text{start}} - 1], \omega[r_{\text{end}} + 1]\}$ is not a valid base pair, and $I_{\text{end}} \leq r_{\text{end}}$. Similarly if $I_{\text{start}} \leq r_{\text{start}}$, then $P_r = P_{\overline{r'}}$ and by Claim 11, $\{\omega[r_{\text{start}} - 1], \omega[\overline{r'}_{\text{end}} + 1]\}$ is not a valid base pair, so $I_{\text{start}} \geq r_{\text{start}}$. ◀

## 5 Empirical relevance of maxStacks and 2D-locked sequences

The aim of this section is to substantiate the promises of the maxStacks model, and our partial algorithmic solution, by evaluating its behavior on random secondary (2D) structures. Our algorithm implementation and experiments are available at `https://doi.org/10.5281/zenodo.20266457`. Namely, we: i) Assess to which extent the maxStacks model represents a good proxy to the Turner model; and ii) Discuss the relevance of the sequences returned by Algorithm 1, based on 2D-locked, on 2D structures.

**Synthetic dataset.** We use a generation algorithm [13] to sample 1960 instances uniformly at random among secondary structures with size ranging from 25 to 175, while ensuring a minimal helix length ranging from 2 to 8 and at least $\theta = 3$ unpaired positions between paired positions. Such instances average 37% unpaired positions, while having relatively short helices, with the average helix size being typically 1.2 times the minimum size.

**Implementation.** Our primary interest is to evaluate the empirical behavior of Algorithm 1, yet the scope of our algorithm is limited to input structures which respect a strict bound on the minimal size of helix. To overcome this limitation, we extend Algorithm 1 into a heuristic called `Locked`$^+$, which functions as follows: if a loop contains helices that are too short to satisfy the Unique Key constraint, we first attempt to *squeeze* the prefix and suffix ($P_r$ and RevComp($P_{\overline{r}}$) in the algorithms) to only retain the first (resp. last) character. If the key cores still do not fit in the remaining space, we replace the substring $\mathsf{AAA}^* k_H$ with a random sequence of length compatible with the actual helix size. These workarounds enable `Locked`$^+$ to produce sequences for any 2D structure without isolated base pairs, while guaranteeing that output sequences are 2D-locked if all helices in the input are sufficiently large.

As a baseline, we also consider `Random`, an algorithm which assigns random compatible nucleotides $\{\{\mathsf{A},\mathsf{U}\},\{\mathsf{G},\mathsf{C}\}\}$ to the individual base pairs of the target, and $\mathsf{C}$ to all unpaired positions, as done within Algorithm 1.

**Benchmarking methodology and metrics.** For each instance, we run both algorithms to produce a single sequence, and report whether or not the produced sequences represent designs for the maxStacks energy model. Namely, we run a $\Theta(n^3)$ dynamic programming algorithm to check if the maxStacks folding of a produced sequence is unique, and coincides with the target. We report in Figure 7 the proportion of maxStacks designs produced by `Locked`$^+$ and `Random`.

Since our final goal is to produce designs under the realistic, yet considerably more intricate, Turner energy model, we evaluate each for uniqueness and optimality under this model as well. We report in Figure 8 the proportion of Turner designs within `Locked`$^+$ and `Random` sequences.

Finally, to assess a refined notion of "quality", we consider the signed energy distance $\Delta\Delta G(\omega, T)$ of a target $T$ to its most stable distant alternative over $\omega$, used in earlier work [3] and defined as

$$\Delta\Delta G(\omega, T) := \min_{T' \neq T} \Delta G(w, T') - \Delta G(\omega, T)$$

with $\Delta G$, the Turner energy in kcal.mol$^{-1}$. Positive values mean that $\omega$ is indeed a design for $T$ in Turner, and is preferred over any alternative by at least $\Delta\Delta G(\omega, T)$ kcal.mol$^{-1}$. On the contrary, a negative $\Delta\Delta G(\omega, T)$ means that, for the sequence, an alternative structure is preferred, and that $\omega$ is not a valid solution to Inverse Folding in the Turner energy model.

Practically, to evaluate $\Delta\Delta G$ in the Turner model, we use tools from the `ViennaRNA` package [10] to produce the energy $\Delta G(T)$ of the target structure $T$ (`RNAeval`), and the first competitive alternative $T' \neq T$ (`RNAsubopt`). We use default options, and only set the maximal energy distance parameter $E$ to 5kcal.mol$^{-1}$ to avoid a combinatorial explosion of suboptimal structures. This implies that $\Delta\Delta G(\omega, T)$ cannot be evaluated to values lower than $-5$ kcal.mol$^{-1}$, so that such sequences achieving this exact value may actually be worse than they appear. We report in Figure 9 the energy distances induced by `Locked`$^+$ and `Random`.

**Results.** First, we note that `Locked`$^+$ sequences largely represent designs in the maxStacks model, even for smaller helix sizes, as seen in Figure 7. This is formally guaranteed for larger helix lengths since `Locked`$^+$ then becomes strictly equivalent to Algorithm 1, and we indeed observe 100% success rate among structures having minimum helix size $\geq 7$. More surprisingly (and promisingly), even for target structures with minimal helix size 4, we observe that 89% of `Locked`$^+$ sequences represent solutions to maxStacks 2D-Inverse Folding

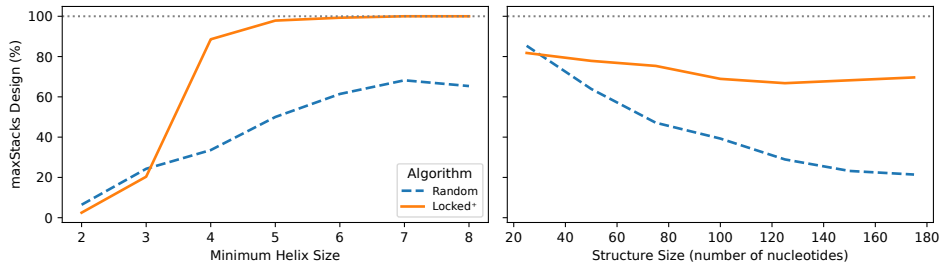



**Figure 7** Proportion of solutions to 2D-Inverse Folding in the maxStacks energy model, for our algorithm (`Locked`$^+$) and a random assignment of base pairs (`Random`), over 1960 test structures.

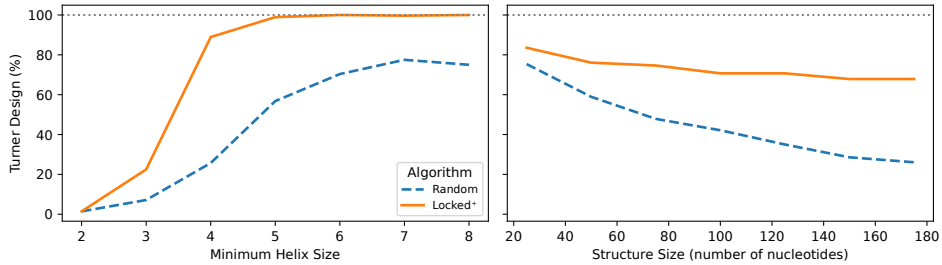



**Figure 8** Proportion of Turner designs within sequences produced by `Locked`$^+$ and `Random` algorithms, for varying size of the smaller helix.

(30% for `Random`). These figures illustrate the fact that the `Locked` condition, while useful towards our partial result, is probably far from necessary except for extreme cases such as outlined by Lemma 1. This also suggests that alternative criteria could be found, and used a foundation for future methods with extended scope of proven validity.

We also remark that `Locked`$^+$ produces promising candidates with respect to the Turner model, as illustrated by Figure 8. Among structures with minimal helix size of 4, 91% directly represent a design in the Turner energy model (99.6% for size 7). In comparison, `Random` contains only 30% of designs in Turner for structures with minimum helix size 4, and 81% for size 7. Moreover, the larger the sequences, the harder it is for a sequence to be a maxStacks or a Turner design. Nonetheless more than 70% of sequences of size 150 are a design in Turner with `Locked`$^+$ against 30% for `Random`, suggesting a better scalability for `Locked`$^+$.

More generally, it appears that the maxStacks model represents a good proxy towards the Turner model, regardless of which algorithm produces maxStacks solutions. Figure 9 partially substantiates this claim: over the whole of 1960 sequences produced by each algorithm (Figure 9, left), 45% of the `Random` and 73% of `Locked`$^+$ sequences, are Turner designs. When restricted to sequences that are designs for maxStacks (Figure 9, right), this success rate increase to more than 85%, both produced by `Random` and `Locked`$^+$. This suggests that the maxStacks model still correlates well enough with the Turner model, and represent a sufficiently good alternative to the Turner model to solve design problems through a combinatorial approach.

Finally, when restricted to maxStacks designs, `Locked`$^+$ not only produces sequences which are more likely valid for Turner, but also represent qualitatively better candidates as shown by the $\Delta\Delta G$ metrics. Indeed, the average $\Delta\Delta G$ is of 1.38 kcal.mol$^{-1}$ for `Locked`$^+$, and 0.55 kcal.mol$^{-1}$ for `Random`. For comparison, a designed sequence with $\Delta\Delta G = 1.38$ is

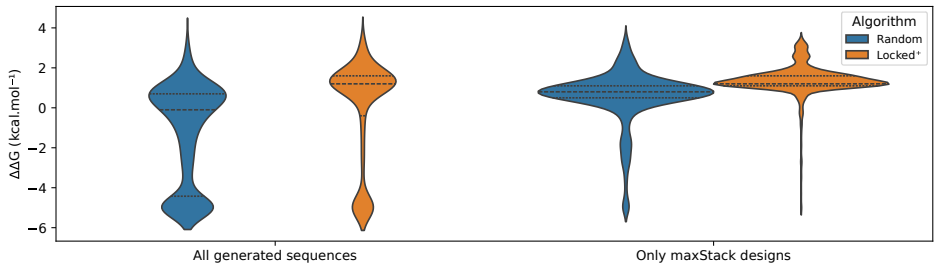



**Figure 9** Signed energy difference ($\Delta\Delta G$), in the Turner model, between the target structure and its best alternative structure. Distributions of $\Delta\Delta G$ for all sequences produced by both `Random` and `Locked`$^+$ (left), and their respective restrictions to maxStacks designs (right). Note that sequences with positive $\Delta\Delta G$ represent Turner designs.

9.4 times more likely to fold, at the thermodynamic equilibrium, into its target structure than into any of its individual competitors (although there may be many such competitors). For $\Delta\Delta G = 0.55$ the ratio of probability drops to 2.4.

## 6 Extending Inverse Folding to include pseudoknotted targets/competitors

In addition to 2D-Inverse Folding, a relevant alternative is to allow the presence of crossing interactions, also called pseudoknots, both in the target structure and competing structures. Note that this is not just a simple generalization of the input structure, since a no-instance in the PK model can become a yes-instance in the 2D model, due to some competitive pseudoknotted structures being disregarded in the 2D version of the problem.

▶ **Problem 2** ((maxStacks) PK-Inverse Folding).
**Input:** *Admissible target structure $T$; length $n$.*
**Output:** *RNA sequence $\omega \in \Sigma^n$, compatible with $T$, such that if $T' \neq T$ is an admissible structure compatible with $\omega$ then $|\mathcal{S}_T| > |\mathcal{S}_{T'}|$; or $\varnothing$ if no such sequence exists.*

Complexity-wise, the 2D-Inverse Folding and PK-Inverse Folding problems differ substantially with respect to the problem of deciding whether a given sequence is actually a solution for a given target structure. This problem requires, given a sequence, to find the structures in which the sequence folds in the maxStacks model. This is solvable in $\mathcal{O}(n^3)$ time in the 2D setting using dynamic programming [8], whereas it becomes NP-hard in the presence of arbitrary crossing/general pseudoknots [11]. Thus, PK-design may not even belong in NP.

**PK-locked sequences and their construction.** With pseudoknots allowed, we cannot avoid easily the rearrangement between two helices from two different loops as done in 2D-locked criteria. To palliate that, we propose to have distinct keys overall instead of distinct keys only per loop. This choice, even if drastic, conveniently simplified proofs, as detailed below, but is also not too excessive in the presence of pseudoknots: any design necessarily features pairwise distinct helices, otherwise two identically designed helices would allow an alternative pseudoknotted structure with the same energy. This leads to the following criteria in the PK setting:

▶ **Definition 3.** *A sequence $\omega$ is* PK-locked *for structure $T$ if $\omega$ is locked, with the following additional condition 4b:*

**4b** ***Unique Keys:*** *$\forall r \in \mathcal{R}$, $\nexists \lambda' \neq \lambda^{\overline{r}}$ such that $\lambda^r$ matches $\lambda'$.*

Note that condition 4b implies the previous conditions 4 and 4a.

▶ **Theorem 12.** *Given a structure $T$, if a sequence $\omega$ is PK-locked, then $\omega$ is a solution of* PK-Design$(T)$.

**Proof.** The proof is the same as for Theorem 3, with Claim 4 being a direct consequence of Lemma 2: any alternative must match keys with their reverse complement in the sequence, but by Condition 4b each reverse complement appears only once and in the desired region. So overall keys are matched together as in the target structure. The proof that helix endpoints are conserved is again the same as in the 2D case. ◀

We then need to adapt Algorithm 1 to produce sequences satisfying condition 4b instead of 4a. This is easily done by assigning distinct keys to all helices in the structure, not only those featured in a common loop. Thus, we modify Line 12 to become:

12: – strings $k_H$ are pairwise distinct

An example of a solution obtained with this adapted version of the algorithm toward pseudoknots is available in Figure 10.

▶ **Theorem 13.** *If $T$ is a pseudoknotted structure over $n$ nucleotides featuring helices of size above $\lceil \log_{3.56}(m) + 6.2 \rceil$, with $m$ number of helices in $T$, then Algorithm 1 with modified Line 12 produces a solution to* PK-Design *in $O(n)$ time and space.*

**Proof.** Similarly to Lemma 8, modified Algorithm 1 does produce an output sequence in linear time. Indeed, the modified line 12 imposes a new constraint, that is clearly satisfiable using the stronger upper bound on the number of helices: size $\lceil \log_{3.56}(m) + 6.2 \rceil$ is sufficient to have distinct keys over all $m$ helices of the target structure (cf Lemma 14 in Appendix).

Furthermore, as in Lemma 9, the output $\omega$ is PK-locked. Indeed, former Line 12 is only used to verify condition 4a, so all other conditions are directly satisfied as in the 2D case. Now for condition 4b, if $\lambda^r$ can be matched to an interval $I$, then as in the 2D case $I = \lambda^{\overline{r'}}$ with $r$ and $r'$ being regions from helices $H$, $H'$ such that $k_H = k_{H'}$. By the modified Line 12, this implies $H = H'$ thus satisfying Condition 4b. ◀

## 7 Conclusion and discussion

We introduced the inverse folding problem in the stacks maximization energy model (maxStacks) and proposed a linear-time algorithm to solve it exactly for sufficiently large helices. We empirically showed why maxStacks represents an interesting compromise between practicality and model expressivity: maxStacks lifts some of the limitations induced by previous propositions and, in particular, enables a design of structures featuring loops of arbitrary size and composition while retaining one of the most important faces from the Turner model. In particular, experimental evaluation of our algorithm indicates that obtained sequences for the secondary structures are promising candidates to be solutions in the Turner energy model. Our algorithmic design relies on a novel property, called locked, which is suitable for any structure "types", including the ones with general pseudoknots. This enabled a linear-time design over challenging sets of structures, due to RNA folding being NP-hard for such structures in maxStacks [11]. Despite unlocking the design in the maxStacks energy

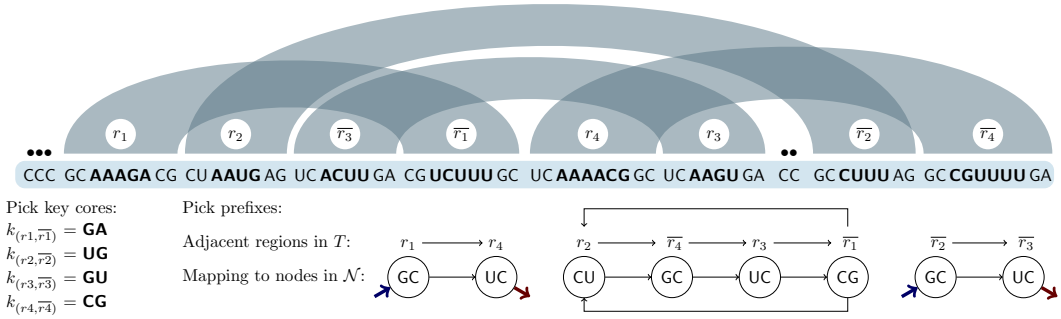



**Figure 10** Execution (top) of Algorithm 1, modified as per Section 6 to support pseudoknots, on a typical pseudoknotted structure (see Figure 6 for a pseudoknot-free example). In this setting, loops are not considered since all key cores are pairwise distinct.

model, some desirable targets still remain beyond the reach of our algorithms. Indeed, even if modestly growing with the max degree (resp. number of helices), the minimal helix length requirement reduces the scope of our algorithms, since they are no longer formally guaranteed to succeed whenever a single helix is too short. We showed that a simple heuristic extension of our algorithm, designing shorter helices as close as possible to what would be put by our algorithm for large enough helices, allows us to still get interesting sequences that are frequently solutions in maxStacks or Turner. We are also confident that future exact extensions of our algorithms can work around this helix size limitation. Plausible directions include an exhaustive enumeration of sequences for helices of "small sizes", ensuring an absence of alternative matchings for regions, possibly leading to a fixed-parameter tractable algorithm in the overall length of small helices. Another extension could consider each bulge or each internal loop as a unique helix, with some fixed unpaired part.

On a complexity level, this work leaves open the possible NP-hardness of inverse folding in the maxStacks model. The closest related hardness result is from Bonnet *et al.* [2], who showed NP-hardness for the maxBPs model with pre-assigned nucleotides, yet left the hardness of the unconstrained version open. When the target structure contains general pseudoknots, given the hardness of folding, we conjecture inverse folding in maxStacks to be coNP-complete. Additionally, the existence of a locked sequence for a given target represents an interesting problem, whose complexity remains open. One could finally consider the problem of finding minimal perturbation (*e.g.* with respect to base pair insertions) such that the modified structure admits a locked sequence. This work already implies that inserting $\mathcal{O}(m \log(\Delta))$ (resp. $\mathcal{O}(m \log(m))$ base pairs is sufficient in the 2D (resp. PK) setting, with $m$ the number of helices and $\Delta$ the max degree of a loop. Moreover, Lemma 1 implies that this bound is tight up to a multiplicative constant. This problem is of interest towards applications, as it is known that minor structural variations (*e.g.* helix extensions) are unlikely to alter biological function, and are often allowed in design methodologies [15].

Finally an interesting extension would be to consider weighted models, each stack having a different real-valued contribution depending on its content. This may prove an important step towards implementing a design algorithm in the full Turner model, where general loops contribute to the energy. In such a setting, Wobble pairs ($\{\mathsf{G}, \mathsf{U}\}$) are typically less stable than others (see Appendix Table 2). Specifically, our algorithm does not use any $\{\mathsf{G}, \mathsf{U}\}$, and implies more stacks in the target than all other structures, even the ones including $\{\mathsf{G}, \mathsf{U}\}$. It follows that our algorithm already solves the design problem in any weighted stack model where stacks without $\{\mathsf{G}, \mathsf{U}\}$ have greater energetic contribution than those including $\{\mathsf{G}, \mathsf{U}\}$.

## References


1 Turner 2004 terminal mismatch parameters. URL: https://rna.urmc.rochester.edu/NNDB/turner04/tm.html.

2 Édouard Bonnet, Paweł Rzążewski, and Florian Sikora. Designing RNA secondary structures is hard. *Journal of Computational Biology*, 27(3):302–316, March 2020. doi:10.1089/cmb.2019.0420.

3 Théo Boury, Samuel Gardelle, Laurent Bulteau, and Yann Ponty. RNA inverse folding can be solved in linear time for structures without isolated stacks or base pairs. *Algorithms for Molecular Biology*, 20(1):20, 2025. doi:10.1186/s13015-025-00278-6.

4 Théo Boury, Leonhard Sidl, Ivo L. Hofacker, Yann Ponty, and Hua-Ting Yao. Old dog, new tricks: exact seeding strategy improves RNA design performances. In Sriram Sankararaman, editor, *Research in Computational Molecular Biology*, pages 134–152. Springer Nature Switzerland, 2025. doi:10.1007/978-3-031-90252-9_9.

5 Robert B. Cary and Gary D. Stormo. Graph-theoretic approach to RNA modeling using comparative data. In Christopher J. Rawlings, Dominic A. Clark, Russ B. Altman, Lawrence Hunter, Thomas Lengauer, and Shoshana J. Wodak, editors, *Proceedings of the Third International Conference on Intelligent Systems for Molecular Biology, Cambridge, United Kingdom, July 16-19, 1995*, pages 75–80. AAAI, 1995. URL: http://www.aaai.org/Library/ISMB/1995/ismb95-009.php.

6 Alexander Churkin, Matan Drory Retwitzer, Vladimir Reinharz, Yann Ponty, Jérôme Waldispühl, and Danny Barash. Design of RNAs: comparing programs for inverse RNA folding. *Briefings in Bioinformatics*, page bbw120, January 2017. doi:10.1093/bib/bbw120.

7 Raymond F. Gesteland, Thomas R. Cech, and John F. Atkins, editors. *The RNA world*. Number 43 in Cold Spring Harbor monograph series. Cold Spring Harbor Laboratory Press, Cold Spring Harbor, NY, 3. ed. edition, 2006.

8 Jozef Haleš, Alice Héliou, Ján Maňuch, Yann Ponty, and Ladislav Stacho. Combinatorial RNA design: designability and structure-approximating algorithm in Watson–Crick and Nussinov–Jacobson energy models. *Algorithmica*, 79(3):835–856, 2017. doi:10.1007/s00453-016-0196-x.

9 Michelle L. Hastings and Adrian R. Krainer. Rna therapeutics. *RNA*, 29(4):393–395, March 2023. doi:10.1261/rna.079626.123.

10 I. L. Hofacker, W. Fontana, P. F. Stadler, L. S. Bonhoeffer, M. Tacker, and P. Schuster. Fast folding and comparison of RNA secondary structures. *Monatshefte für Chemie / Chemical Monthly*, 125(2):167–188, 1994. Number: 2. doi:10.1007/BF00818163.

11 Rune B. Lyngsø. *Complexity of pseudoknot prediction in simple models*, pages 919–931. Springer Berlin Heidelberg, 2004. doi:10.1007/978-3-540-27836-8_77.

12 David H. Mathews, Matthew D. Disney, Jessica L. Childs, Susan J. Schroeder, Michael Zuker, and Douglas H. Turner. Incorporating chemical modification constraints into a dynamic programming algorithm for prediction of RNA secondary structure. *Proceedings of the National Academy of Sciences*, 101(19):7287–7292, 2004. Publisher: Proceedings of the National Academy of Sciences. doi:10.1073/pnas.0401799101.

13 Yann Ponty. *Ensemble algorithms and analytic combinatorics in RNA bioinformatics and beyond*. Habilitation à diriger des recherches, Université Paris-Saclay, May 2020. URL: https://theses.hal.science/tel-03219977.

14 Erik Poppleton, Niklas Urbanek, Taniya Chakraborty, Alessandra Griffo, Luca Monari, and Kerstin Göpfrich. Rna origami: design, simulation and application. *RNA Biology*, 20(1):510–524, July 2023. doi:10.1080/15476286.2023.2237719.

15 Matan Drory Retwitzer, Vladimir Reinharz, Alexander Churkin, Yann Ponty, Jérôme Waldispühl, and Danny Barash. Incarnafbinv 2.0: a webserver and software with motif control for fragment-based design of RNAs. *Bioinformatics*, 36(9):2920–2922, January 2020. doi:10.1093/bioinformatics/btaa039.

16 Siyu Wang, Drew Weissman, and Yizhou Dong. Rna chemistry and therapeutics. *Nature Reviews Drug Discovery*, 24(11):828–851, July 2025. doi:10.1038/s41573-025-01237-x.

17 Hua-Ting Yao, Cédric Chauve, Mireille Regnier, and Yann Ponty. Exponentially few RNA structures are designable. In *Proceedings of the 10th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics*, BCB '19, pages 289–298. ACM, September 2019. doi:10.1145/3307339.3342163.

18 Jeffrey Zuber, Susan J Schroeder, Hongying Sun, Douglas H Turner, and David H Mathews. Nearest neighbor rules for RNA helix folding thermodynamics: improved end effects. *Nucleic Acids Research*, 50(9):5251–5262, May 2022. doi:10.1093/nar/gkac261.


# A Maximum loop size of 2D structures depending on helix size that can be solved with our Algorithm 1

**Table 1** Upper bound on the maximum loop size in the 2D structure $m$ as a function of the available helix size in the target structure, obtained with Lemma 14 (in Appendix).

| Helix size $\geq$ | 7 | 8 | 9 | 6+k | $\lceil \log_{3.56}(m) + 6.2 \rceil$ |
|---|---|---|---|---|---|
| Design structures with $m \leq$ | 3 | 9 | 35 | $\frac{\sqrt{17}-1}{4}\left(\frac{\sqrt{17}+3}{2}\right)^k$ | $(\simeq 0.78 * (3.56)^k)$ |

# B Energy contributions of different base pairs

**Table 2** Average energy of stacks using any two base pairs in the Turner model [12], disregarding uncertainty of measures (values taken from [1]). Averages are taken over all possible configurations of the two base pairs. Note that replacing any $\{\mathsf{A},\mathsf{U}\}$ or $\{\mathsf{G},\mathsf{C}\}$ base pair by $\{\mathsf{G},\mathsf{U}\}$ increases the energy (thus reducing the stability). This overall suggests that the local stability of the stack, and thus the global stability of the structure, is degraded by the presence of $\{\mathsf{G},\mathsf{U}\}$.

| Type of stacks | $\{\{\mathsf{G},\mathsf{C}\},\{\mathsf{G},\mathsf{C}\}\}$ | $\{\{\mathsf{A},\mathsf{U}\},\{\mathsf{G},\mathsf{C}\}\}$ | $\{\{\mathsf{A},\mathsf{U}\},\{\mathsf{A},\mathsf{U}\}\}$ |
|---|---|---|---|
| Energy (kcal/mol) | -3,075 | -2,195 | -1,0725 |
| Type of stack | $\{\{\mathsf{G},\mathsf{C}\},\{\mathsf{G},\mathsf{U}\}\}$ | $\{\{\mathsf{A},\mathsf{U}\},\{\mathsf{G},\mathsf{U}\}\}$ | $\{\{\mathsf{G},\mathsf{U}\},\{\mathsf{G},\mathsf{U}\}\}$ |
| Energy (kcal/mol) | -1,89 | -1,045 | +1 |

# C Proofs for Section 2 (Problems statement, definitions and motivation)

**Proof of Lemma 1.** We simply show that the loop of size $h < \log_6(\Delta)$, i.e. the loop in $T$ contains more than $6^h$ helices. By contradiction, suppose that there exists a design $\omega$ of $T$. As there are no more than $6^h$ assignment possibilities for the helices of size $h$ in the loop (due to the only 6 possible choices of base pairs), there exists necessarily two helices $H_1 = (r_1, \overline{r}_1)$ and $H_2 = (r_2, \overline{r}_2)$ such that $\omega[r_1] = \omega[r_2]$ and $\omega[\overline{r}_1] = \omega[\overline{r}_2]$. In that case, one can construct $T'$ containing the same helices as $T$, with the exception of two helices $H'_1 = (r_2, \overline{r}_1)$ and $H'_2 = (r_1, \overline{r}_2)$ replacing $H_1$ and $H_2$ respectively. Note that $H'_1$ and $H'_2$ are indeed feasible (forming no new crossing) since $r_1 = r_2$ and $\overline{r}_1 = \overline{r}_2$. In addition, $H'_1$ and $H'_2$ feature as many stacks as $H_1$ and $H_2$, making $T'$ a competitive alternative in maxStacks. ◀

## D Proofs for Section 4 (Producing locked sequences for targets with sufficiently large helices)

▶ **Lemma 14.** *A* valid key core *is a string over* $\{A, U, C, G\}$ *starting with* $\{U, G, C\}$ *and avoiding* $AA$ *and* $UU$*. For any* $k$ *and* $m \leq \frac{\sqrt{17}-1}{4}\left(\frac{\sqrt{17}+3}{2}\right)^k$ *(or* $k = 1$ *and* $m \leq 3$*), there exist at least* $m$ *distinct valid key cores of length* $k$*. Conversely, for any* $m$*, there exist* $m$ *distinct valid key cores of length* $k$ *for any* $k \geq \log_{3.56}(m) + 0.2$*.*

**Proof.** In the following we write $r$ and $c$ for the positive roots of the equations $c = 2r + 2$ and $r + 2 = rc$, i.e. $c = \frac{\sqrt{17}+3}{2} \simeq 3.56$ and $r = \frac{\sqrt{17}-1}{4} \simeq 0.78$. We write $m_x(k)$ for the number of length-$k$ strings starting with $x \in \{\mathsf{A}, \mathsf{U}, \mathsf{G}, \mathsf{C}\}$ and avoiding $\mathsf{AA}$ and $\mathsf{UU}$. We show by induction that:

$$m_{\mathsf{A}}(k) = m_{\mathsf{U}}(k) \geq rc^{k-1} \quad \text{and} \quad m_{\mathsf{C}}(k) = m_{\mathsf{G}}(k) \geq c^{k-1}$$

Indeed, $m_{\mathsf{A}}(k) = m_{\mathsf{U}}(k)$ and $m_{\mathsf{G}}(k) = m_{\mathsf{C}}(k)$ are clear by symmetry. For $k = 1$, $m_{\mathsf{A}}(k) = 1 \geq rc^0$ and $m_{\mathsf{G}}(k) = 1 = c^0$. Also, we have the following recursive relations :

$$\begin{aligned} m_{\mathsf{A}}(k+1) &= m_{\mathsf{U}}(k) + m_{\mathsf{G}}(k) + m_{\mathsf{C}}(k) && \geq (r+2)c^{k-1} && = rc^k \\ \text{and } m_{\mathsf{G}}(k+1) &= m_{\mathsf{A}}(k) + m_{\mathsf{U}}(k) + m_{\mathsf{G}}(k) + m_{\mathsf{C}}(k) && \geq (2r+2)c^{k-1} && = c^k \end{aligned}$$

Thus, the number of valid key cores is at least $m_{\mathsf{U}}(k) + m_{\mathsf{C}}(k) + m_{\mathsf{G}}(k) = m_{\mathsf{A}}(k+1) = rc^k$. Finally, the logarithmic bound is obtained as follows: for $k \geq \log_{3.56}(m) + 0.2$, using $c \geq 3.56$ and $\log_c(r) \geq -0.2$ we have $k \geq \log_c(m) - \log_c(r)$ so $rc^k \geq m$: there are indeed $m$ distinct valid keys of size $k$. ◀

**Proof of Lemma 8.** We show that the algorithm can always find nodes $P_r$ satisfying the criteria lines 5 to 8, and key cores satisfying the criteria lines 10 to 12.

Regarding nodes, we consider the graph $\mathcal{G}$ of adjacent successor/predecessors: use one node per region of $\mathcal{R}_T$, and add an arc between $r$ and $r'$ if $\overline{r}_{\text{end}} + 1 = r'_{\text{start}}$ (equivalently, $r'$ is the successor of $r$ and $\overline{r}, r'$ are adjacent). Note that each region has in-degree at most one and out-degree at most one, so it partitions into paths and cycles (subsets of loops). Moreover, if $r$ has out-degree 0 then $\overline{r}_{\text{end}} + 1$ is unpaired (or $> n$), and if $r$ has in-degree 0 then $r_{\text{start}} - 1$ is unpaired (or $< 1$). Thus, lines 5 to 7 are satisfied if each connected component of $\mathcal{G}$ can be mapped to a (not necessarily simple) path or cycle of the length in $\mathcal{N}$, where nodes with in-degree 0 (resp. out-degree 0) are mapped to enter-nodes (resp. exit-nodes) of $\mathcal{N}$. This is always possible by Observations 7.9 and 7.10. Finally, satisfying line 8 is somewhat more technical. We show that nodes can be assigned to regions sequentially such that, at each step, there is at most one region assigned $\mathsf{CU}$ whose complement does not have any assigned node. Such a region $r$ with $P_r = \mathsf{CU}$ and $P_{\overline{r}}$ still undefined is hereafter called a *single CU-region*. This is achieved by picking the connected component containing the complement of a single $\mathsf{CU}$-region (if any, otherwise pick any component). If the component is a path, by Observation 7.9 we assign non-$\mathsf{CU}$ nodes to all regions: there is no single $\mathsf{CU}$-region after such step. If the component is a cycle, we assign nodes so that at most one node $r$ of the cycle is assigned $\mathsf{CU}$ with Observation 7.10 ($r$ is any other region than the complement of the previous $\mathsf{CU}$-region); there is at most one single $\mathsf{CU}$-region at this point.

Regarding key cores, let $\ell = \lceil \log_{3.56}(m) + 0.2 \rceil$. By Lemma 14, key cores can be built with length $\leq \ell$, and assigned successively to helices. Condition line 12 can easily be satisfied by assigning key cores in a "top-down" ordering of loops: first from the outer loop, then in the intervals inside each helix recursively, so that at most one helix in each loop has an

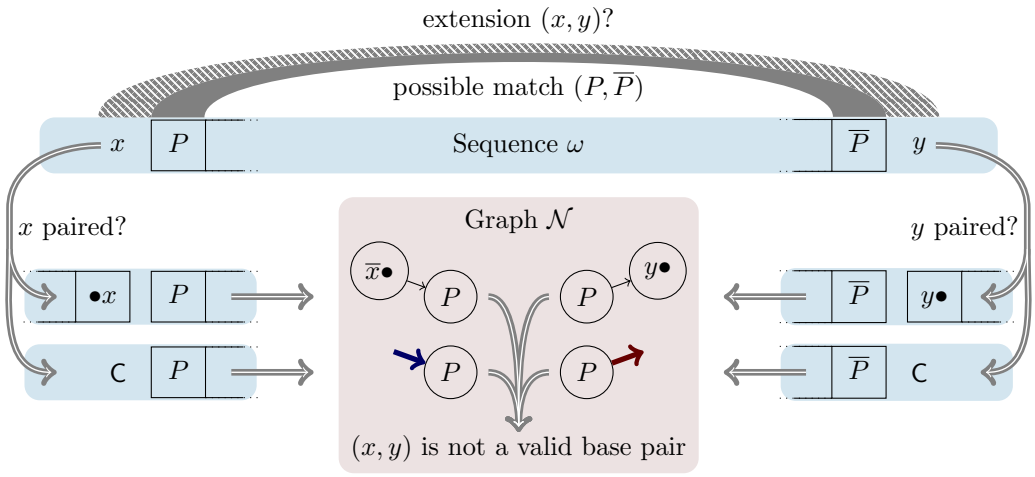



**Figure 11** Illustration of Claim 11, providing the core of the non-extension condition: if the prefix of a region matches the suffix of another (including with both regions of a helix, but not necessarily), then the match cannot be extended further beyond the region border, even with $\{\mathsf{G}, \mathsf{U}\}$ pairs. Here we write $\overline{P}$ for RevComp($P$), and $\bullet$ is used as a wildcard for any base, i.e. $y\bullet$ is a $y$-node of $\mathcal{N}$.

already assigned key core and all other helices can be assigned the $m-1$ available strings. Overall each key core has length $\leq \ell$, and lines 16 and 18 design helices with strings of size at least $\ell + 6$, which corresponds to the required bound on the helix size.

The overall algorithm indeed takes linear time and space in the length of the sequence $n$: paths and cycles in $\mathcal{G}$ have a cumulative size of 2 times the number of helices plus the number of unpaired positions, which can be enumerated in $O(n)$. Paths and cycles and next assigned pre-selected paths in $\mathcal{N}$ based on their lengths (the constraint over nodes $\mathsf{CU}$ only forces the components to be visited in the right order, without affecting the overall complexity). Then, key cores can also be computed in linear time in the total number of helices. The letters assigned to each helix are finally some copies of extremity assignments and keys in linear time of the helix length. For the space complexity, extremities are not stored as assigned on the fly when enumerated. $\mathcal{N}$ is stored in $O(1)$ and the cumulative size of the stored keys is $O(n)$. ◀

Proof of Claim 10. Such patterns cannot be included in strings $P_r$, $P_{\overline{r}}$ or their reverse complements (nodes or $\mathcal{N}$ are neither $\mathsf{AA}$ nor $\mathsf{UU}$), nor in $k_r$, $\overline{k_r}$ (by line 11). They cannot overlap two such parts: $P_r$ does not end with $\mathsf{A}$ (Observation 7.2), $k_r$ does not start with $\mathsf{A}$, and $P_{\overline{r}}$ does not end with $\mathsf{U}$ and RevComp($P_{\overline{r}}$) does not start with $\mathsf{A}$ nor $\mathsf{U}$ (using Observation 7.2 and $P_{\overline{r}} \neq \mathsf{CU}$). So patterns $\mathsf{AA}$ and $\mathsf{UU}$ cannot be included in a single region (beside $\mathsf{A}^i$ and $\mathsf{U}^i$ in the reverse complement). They cannot use unpaired positions either (since such positions receive a $\mathsf{C}$, as per line 3), and they cannot either use two endpoints of consecutive regions. Indeed, if $\omega[r_{\text{end}}] = \omega[r'_{\text{start}}] = \mathsf{A}$ (the proof is similar for $\mathsf{U}$) for any two regions $r, r'$, then $P_{r'}$ is an $\mathsf{A}$-node (since $P_{r'}$ starts with $\mathsf{A}$) and $P_{\overline{r}}$ is a $\mathsf{U}$-node (since RevComp($P_{\overline{r}}$) ends with $\mathsf{A}$). By Observation 7.3 there is no arc from $P_{\overline{r}}$ to $P_{r'}$ and by line 5, $r_{\text{end}} \neq r'_{\text{start}} - 1$, so these two regions do not form any $\mathsf{AA}$ pattern. ◁

Proof of Claim 11. See Figure 11. Write $P = P_r = P_{r'}$, $x = \omega[r_{\text{start}} - 1]$ and $y = \omega[\overline{r'}_{\text{end}} + 1]$.

If $r_{\text{start}} - 1$ is an unpaired position, then $x = \mathsf{C}$ and $P_r$ is an enter-node (by line 6). Otherwise, $r_{\text{start}} - 1$ is the last position of a region $q$ with $P_{\overline{q}} \to P$ (line 5), and $X = P_{\overline{q}}$ is an $\overline{x}$-node in $\mathcal{N}$. Similarly, $\overline{r'}_{\text{end}} + 1$ is either unpaired so $y = \mathsf{C}$ and $P = P_{r'}$ is an exit-node (line 7), or it is paired to a region $q$ where $Y = P_q$ is a $y$-node and $P \to Y$ in $\mathcal{N}$.

Overall, if both are unpaired then $x = y = \mathsf{C}$; if only $r_{\text{end}} + 1$ is unpaired then $P$ is an exit-node and by Observation 7.8, $y = \mathsf{C}$ and $\overline{x} \neq \mathsf{C}$ so $\{x, y\}$ is not valid. If only $\overline{r'}_{\text{start}} - 1$ is unpaired then $P$ is an enter-node and by Observation 7.7, $x = \mathsf{C}$ and $y \neq \mathsf{G}$, so $\{x, y\}$ is not valid. Otherwise we have $X \to P \to Y$ in $\mathcal{N}$ with $X$ an $\overline{x}$-node and $Y$ a $y$-node. By Observation 7.4, $\overline{x} \neq y$ so $\{x, y\}$ is neither $\{\mathsf{A}, \mathsf{U}\}$ nor $\{\mathsf{G}, \mathsf{C}\}$. By Observation 7.5 $(x, y) \neq (\mathsf{G}, \mathsf{U})$ and by Observation 7.6, $(x, y) \neq (\mathsf{U}, \mathsf{G})$. Overall $\{x, y\}$ is not a valid base pair. ◁

# Revisiting $O(n \log \log n)$ Chaining for Anchored Edit Distance

**Nicola Rizzo**
University of Helsinki, Finland

**Ragnar Groot Koerkamp**
Karlsruhe Institute of Technology, Germany

## Abstract

Colinear chaining is a classical heuristic for sequence alignment: it enables scalable genome comparison and is a main component of many state-of-the-art read mappers based on seed-chain-extend. The earliest $O(n \log \log n)$ and $O(n \log n)$ time algorithms by Eppstein et al. (J. ACM, 1992) chained $n$ fragments between two sequences $T$ and $Q$ while minimizing a gap cost based on the diagonal distance $\Delta_{\text{diag}}$ between consecutive fragments. They also forbid fragment overlaps, which are essential in current chaining formulations: in long-read mapping, overlaps improve sensitivity and avoid restrictions on the fragment class considered. Jain, Gibney, and Thankachan (J. Comput. Biol. 2022) recently combined a $\Delta_{\text{diag}} = |\Delta_T - \Delta_Q|$ overlap cost with the classic $L_\infty = \max(\Delta_T, \Delta_Q)$ gap cost that takes the maximum between the horizontal and vertical gap between the fragments and they proved that chaining under this cost model is equivalent to the *anchored edit distance*.

We improve the existing $O(n \log^3 n)$-time algorithm for anchored edit distance to $O(n \log \log n)$ time in $O(n)$ space, by combining the gap-cost computation of Chao and Miller (Algorithmica, 1995) with the overlap-cost computation of Baker and Giancarlo (ESA, 1998). By developing `llchain`, a simpler $O(n \log n)$-time implementation of our method, we show how chaining algorithms that might have been recently overlooked by the bioinformatics community scale competitively to millions of fragments and large genomes. On average, `llchain` is 10× faster than other methods on instances with 3 000 000 anchors, and over 2.3× faster on MEMs between HiFi reads and a reference human genome.

**2012 ACM Subject Classification** Theory of computation → Design and analysis of algorithms; Theory of computation → Sorting and searching; Applied computing → Genomics

**Keywords and phrases** Colinear chaining, Anchored edit distance, Sequence alignment, Predecessor structure

**Digital Object Identifier** 10.4230/LIPIcs.WABI.2026.13

**Supplementary Material** *Software*: `https://github.com/nrizzo/llchain` [41]
archived at `swh:1:dir:78de8057c7f61afc5518ef3b8a46a5ac91e21d52`

**Funding** *Nicola Rizzo*: European Union's Horizon Europe research and innovation programme under grant agreement No 101060011 (TeamPerMed).

**Acknowledgements** The authors wish to thank the Finnish Computing Competence Infrastructure (FCCI) for supporting this project with computational and data storage resources.

## 1 Introduction

Colinear chaining, known throughout the decades as *Wilbur–Lipman sequence comparison*, *global fragment chaining*, *sparse dynamic programming*, and just *chaining* is a fundamental technique for approximate alignment that dates back to the 1980s [45, 46, 18, 32]. Chaining techniques have been widely adopted in different sub-fields of computational genomics: whole-genome alignment tools [33]; sequence aligners [23, 24, 16]; read mappers employing the seed-chain-extend strategy [42]; and even assembly evaluation [7, 27]. While the general chaining problem statement is consistent in the literature – to find the highest-scoring ordered


© Nicola Rizzo and Ragnar Groot Koerkamp;
licensed under Creative Commons License CC-BY 4.0



subset of the input fragments that is consistent with an alignment – the exact formulation and algorithms change depending on the application, and it can be difficult to compare the different results. In particular, the behaviour of chaining is mainly affected by fragment score, gap cost, and overlap cost. Popular aligners use a heuristic chaining cost based on the diagonal distance and allow overlaps [24, 37, 20], whereas some methods chain without overlaps and consider a function of the gaps in the two sequences [1, 34], in one case obtaining a *lower bound* on the cost of the optimal alignment [16]. Instead, we consider the *anchored edit distance* [3, 25, 21], which gives an *upper* bound on the true edit distance. In this model, a number of *anchors* or *fragments* are given, which are exact matches between the two input sequences. Then, the anchored edit distance is the minimum cost of a global alignment where only the matches supported by an anchor are free.

▶ **Definition 1** (Anchored edit distance). *Given two strings $T$ and $Q$, an* anchor *(or* fragment*) $a = (t_s, t_e, q_s, q_e)$ is a tuple for which $T[t_s, t_e) = Q[q_s, q_e)$. Given an arbitrary set of $n$ anchors, the* anchored edit distance *is the cost of a global alignment of $T$ and $Q$ that minimizes the total number of insertions, deletions, substitutions, and matches* not supported by an anchor*. Thus, only* supported *matches are free.*

Jain et al. [21] show that the anchored edit distance can be found by computing a minimum-cost chain using an $O(n \log^3 n)$ algorithm, and Rizzo et al. [39] simplified the chaining cost as follows.

▶ **Theorem 2** (Equivalence to chaining). *The anchored edit distance equals the minimum-cost chain of fragments under the following colinear chaining cost between the end $E = (t_e, q_e)$ of one anchor $a$ and the start $S' = (t'_s, q'_s)$ of the next anchor $a'$:*

$$\mathsf{connect}(a, a') := \max(\mathrm{L}_\infty(E, S'), \Delta_{\mathrm{diag}}(E, S')),$$

*where*

$$\mathrm{L}_\infty((t,q),(t',q')) := \max(0, \Delta_T, \Delta_Q) = \max(0, t'-t, q'-q) \quad \textit{is the maximum gap, and}$$
$$\Delta_{\mathrm{diag}}((t,q),(t',q')) := |\Delta_T - \Delta_Q| = |\mathrm{diag}(t,q) - \mathrm{diag}(t',q')| \quad \textit{is the diagonal distance.}$$

**Contributions.** Our main contribution is to connect the anchored edit distance to previously studied chaining problems, resulting in an $O(n \log \log n)$ theoretical algorithm working in $O(n)$ space. Our method works by combining the $\mathrm{L}_\infty$ gap-cost computation in $O(n \log \log n)$ time of Eppstein et al. [14] and Chao and Miller [9] with the $\Delta_{\mathrm{diag}}$ overlap cost computation of Baker and Giancarlo [3]. The $O(n \log \log n)$ time complexity is obtained by using a fast integer sorting algorithm as well as predecessor structures. A simplified $O(n \log n)$ time implementation, `llchain`, performs up to 10× faster (2.3× on average) than the existing solutions for anchored edit distance, when chaining the maximal exact matches (MEMs) of length 50 or more between HiFi reads and a human genome.

## 2 Related work

To provide the needed historical context to our work and to help the reader navigate the literature, we start with a review of the most important results on pairwise chaining focusing on the time complexity in the number of input anchors, summarized in Table 1.

The original chaining formulations by Wilbur and Lipman [46] considered as input exact-match fragments between two sequences $T$ and $Q$, to be selected and connected together in an order that is consistent with an alignment (i.e. a *chain*) while maximizing the number

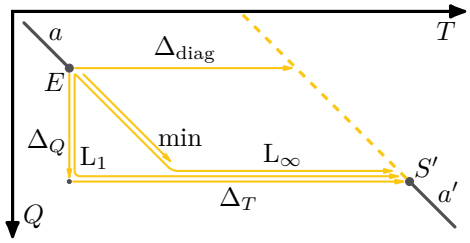



**Figure 1** Different possible chaining costs between two anchors $a$ and $a'$: the distance is between the end $E$ of $a$ and start $S'$ of $a'$, and can be the distance in $Q$ ($\Delta_Q$), the distance in $T$ ($\Delta_T$), or the sum ($L_1$), minimum (min), maximum ($L_\infty$), or absolute difference ($\Delta_{\text{diag}} = |\Delta_T - \Delta_Q|$) of these.

of matched bases minus a constant cost for each gap [45], or maximizing *weighted* matches minus an arbitrary gap cost $g(\Delta_T, \Delta_Q)$ [46], where $\Delta_Q$ and $\Delta_T$ are the lengths of the gaps in $T$ and $Q$ (see Figure 1). They provided an $O(n^2)$ time algorithm. In 1992, the first efficient sub-quadratic-time algorithms were developed by Eppstein et al. [14, 15] to maximize the number of matched bases minus gap costs computed as $f(\Delta_{\text{diag}})$, a one-dimensional function in the diagonal distance between consecutive fragments, and they run in $O(n \log\log n)$ or $O(n \log n)$ time, depending on whether $f$ is linear or concave. Gap costs of the form $f(\Delta_{\text{diag}})$ are still widely used today [24, 37] but they do not explicitly penalize large gaps. This was addressed in efficient algorithms in 1995[1] by Myers and Miller [29] when generalizing chaining to two or more sequences, and by Chao and Miller [9] when extending the approach by Eppstein et al. to find $m$ best-scoring local alignments. Myers and Miller's gap cost $\varepsilon \cdot \min(\Delta_T, \Delta_Q) + \lambda \cdot \Delta_{\text{diag}}$ found a lot of attention in the alignment of whole genomes [33], whereas pairwise chaining implementations like `CHAINER` [1] concentrated on the $L_1 = \Delta_T + \Delta_Q$ gap cost (i.e. $(\varepsilon, \lambda) = (2, 1)$). To the best of our knowledge, the first chaining implementation to support generic $\varepsilon$ and $\lambda$ – and thus $L_\infty = \max(\Delta_T, \Delta_Q)$ for $(\varepsilon, \lambda) = (1, 1)$ – is `clasp` [34]. All chaining formulations before 1998, and many formulations since, do not allow anchors to overlap (see e.g. Figure 2): historically overlaps have been ignored, removed in a pre-processing step, or recovered in downstream refinement phases [10, 8]. However, overlaps improve sensitivity and are widely employed in long-read alignment, where the chaining formulation of `minimap2` [24] is the *de facto* standard way of chaining minimizer seeds, a popular class of sampled fixed-length seeds [42]. Other notable applications of chaining include the evaluation of genome assemblies [7, 27], chaining with inversions [37], and computing an admissible lower bound for the A*PA heuristic to sequence alignment [16].

A complementary problem is that of *anchored longest common subsequence* (anchored LCS), where one searches an alignment that maximizes the number of matches supported by an anchor. Baker and Giancarlo [3] showed in 1998 that chaining to minimize an $L_1 = \Delta_T + \Delta_Q$ gap cost and a $\Delta_{\text{diag}}$ overlap cost actually solves the anchored LCS, and solved the problem in $O(n \log\log n)$ time. The connection to LCS was rediscovered in 2021 by Mäkinen and Sahlin [25] with an alternative but equivalent chaining formulation. Further work in this direction includes LCS$k$ [6, 5], where anchors are the set of all $k$-mers (exact matches of length $k$) and may not overlap, and LCS$k$+ [36], where anchors are all matches of length at least $k$. In [35], an $O(n \log \ell)$ algorithm is given, where $\ell \leq n$ is the number of anchors in the optimal chain.

[1] Zhang et al. [48] define an affine gap score for chaining fragments between $k \geq 2$ sequences already in 1994, but they address the problem with $k$-dimensional trees and do not provide a worst-case complexity analysis.

**Table 1** Main algorithms and implementations (in `typewriter font`) for the global or semi-global chaining of $n$ fragments between two sequences $T$ and $Q$, with $Q$ the smaller string. For fragment types, "exact" refers to exact matches, "square" to matches of the same length, and "inexact" to approximate matches. For "arbitrary" fragment scores, each fragment has a score in input, whereas "matches" is equal to the number of matching bases in the corresponding alignment (also known as mutual coverage [25]). For the gap costs, $f(\Delta_T, \Delta_Q)$ is an arbitrary 2D function, $\ell(\Delta_{\text{diag}})$ is a linear 1D, $f_\cup$ is a concave function, $p_\cup$ is a piece-wise function with a concave part flanked by constant values, $\varepsilon$ and $\lambda$ are linear substitution and indel costs, and $\mathrm{L}_\infty = \max(\Delta_T, \Delta_Q)$. Gap costs that have $\mathrm{L}_\infty$ as a special case are marked in yellow, and $\Delta_{\text{diag}}$ overlap costs in blue.

| chaining formulation | fragment type | fragment score | gap cost | overlap cost | time complexity |
|---|---|---|---|---|---|
| Wilbur and Lipman, 1983 [45] | exact | length | constant | – | $O(n^2)$ |
| Wilbur and Lipman, 1984 [46] | square | context-dependent | $f(\Delta_T, \Delta_Q) \geq 0$ | – | $O(n^2)$ |
| Eppstein et al., 1992 [14] | exact | matches | $\ell(\Delta_{\text{diag}})$ | $\Delta_{\text{diag}} = 0$ penalty | $O(n \log \log n)$ |
| Eppstein et al., 1992 [15] | exact | matches | $f_\cup(\Delta_{\text{diag}})$ | $\Delta_{\text{diag}} = 0$ penalty | $O(n \log n)$ |
| Myers and Miller, 1995 [29] | inexact | arbitrary | $\varepsilon \cdot \min(\Delta_T, \Delta_Q) + +\lambda \cdot \lvert\Delta_T - \Delta_Q\rvert$ | – | $O(n \log^2 n)$ |
| Chao and Miller, 1995 [9] | exact | length | affine | – | $O(n \log \log n)$ |
| Baker and Giancarlo, 1998 [3] | exact | – | $L_1 = \Delta_T + \Delta_Q$ | $\Delta_{\text{diag}}$ | $O(n \log \log n)$ |
| Abouelhoda and Ohlebusch, 2004 [1, 2] `CHAINER` | inexact | arbitrary | $L_1$ | – | $O(n \log n)$ |
| Abouelhoda and Ohlebusch, 2005 [2] | inexact | arbitrary | $\varepsilon \cdot \min(\Delta_T, \Delta_Q) + +\lambda \cdot \lvert\Delta_T - \Delta_Q\rvert$ | – | $O(n \log n \log \log n)$ |
| Otto et al., 2011 [34] `clasp` | inexact | arbitrary | $\varepsilon \cdot \min(\Delta_T, \Delta_Q) + +\lambda \cdot \lvert\Delta_T - \Delta_Q\rvert$ | – | $O(n \log^2 n)$ |
| Bushmanova et al., 2016 [7] `rnaQUAST` | inexact | arbitrary | – | $\mathsf{ov}_T$ | $O(n^2)$ |
| Li, 2018 [24] `minimap2` | fixed-length, exact | matches | $p_\cup(\Delta_{\text{diag}})$ | $p_\cup(\Delta_{\text{diag}})$ | $O(n^2)$ |
| Sahlin and Mäkinen, 2020 [25] | exact | matches | 0 | 0 | $O(n \log n)$ |
| Ren and Chaisson, 2021 [37] `lra` | inexact | arbitrary | $f_\cup(\Delta_{\text{diag}})$ | – | $O(n \log^2 n)$ |
| Sahlin and Mäkinen, 2021 [25, 43] `uLTRA` | inexact | mutual, weighted coverage | $\Delta_Q$ | $\varepsilon \cdot \max(\mathrm{ov}_T, \mathrm{ov}_Q)$ | $O(n^2)$ |
| Jain et al., 2022 [21, 39] | exact | – | $\mathrm{L}_\infty$ | $\Delta_{\text{diag}}$ | $O(n \log^3 n)$ |
| Groot Koerkamp and Ivanov, 2024 [16] `A*PA` | non-overlapping, inexact | edit distance | $\max(\Delta_{\text{diag}}, \lfloor \Delta_T / k \rfloor)$ | – | $O(n \log^2 n)$ |
| **this work**, Alg 1 | exact | – | $\mathrm{L}_\infty$ | $\Delta_{\text{diag}}$ | $O(n \log \log n)$ |
| **this work**, `llchain` | exact | – | $\mathrm{L}_\infty$ | $\Delta_{\text{diag}}$ | $O(n \log n)$ |

## 3 Preliminaries on chaining, range minimum queries, priority queues

We adapt the notation from Rizzo, Cáceres, and Mäkinen [39] to right-open intervals and 0-indexed strings. We denote integer interval $\{x, x+1, \ldots, y\}$ as $[x..y]$, or $[x..y+1)$. Given 0-indexed strings $T$ and $Q$ over finite alphabet $\Sigma$, interval pair $a = ([t_s..t_e), [q_s..q_e))$ is an *exact match anchor* (or fragment, or just anchor) between $T$ and $Q$ if $T[t_s..t_e) = Q[q_s..q_e)$, where $0 \le t_s \le t_e \le |T|$ and $0 \le q_s \le q_e \le |Q|$ (empty anchors are used as sentinels at the start and end of the text). The main invariant of exact match anchors that we will sometimes use is $t_e - t_s = q_e - q_s \iff t_e - q_e = t_s - q_s$.

A colinear chaining formulation is first defined via *anchor precedence*, a partial order $\prec$ specifying which anchors can follow one another to compose a chain $a_1 \prec \cdots \prec a_c$. One then maximizes the *score* of a chain. Indeed, all chaining formulations in Table 1 maximize a formula of the type $\mathsf{score}(A) = \sum_{i=1}^{c} \mathsf{fscore}(a_i) - \sum_{i=1}^{c-1} \mathsf{connect}(a_i, a_{i+1})$, where $\mathsf{fscore}(a)$ is the non-negative fragment score obtained by picking anchor $a$, and $\mathsf{connect}(a_i, a_{i+1})$ is the non-negative penalty (cost) of choosing $a_{i+1}$ after $a_i$. Since our chaining formulation will have no fragment score, our goal will be to minimize a function which is very similar to $\mathsf{cost}(A) = \sum_{i=1}^{c-1} \mathsf{connect}(a_i, a_{i+1})$, the cost of connecting the anchors in a chain.

### 3.1 Full problem statement

Consider the following definitions alongside Figure 2.

▶ **Definition 3** (Anchor precedence and colinear chain [3, 21]). *Given $a = ([t_s..t_e), [q_s..q_e))$ between $T$ and $Q$, let $\mathsf{start}(a) = (t_s, q_s)$ be the start-point of $a$. Then $a$* precedes *$a' = ([t'_s..t'_e), [q'_s..q'_e))$, in symbols $a \prec a'$, if $\mathsf{start}(a)$ is distinct from and reaches $\mathsf{start}(a')$ in a possible alignment, that is,*

$$a \prec a' \quad \text{iff} \quad t_s \le t'_s \text{ and } q_s \le q'_s \text{ and } (t_s, q_s) \ne (t'_s, q'_s).$$

*Then, a sequence of anchors $a_1, \ldots, a_c$ is called a* colinear chain *or just $\prec$-chain if $a_i \prec a_{i+1}$ for all $i \in [1..c)$. Note that relation $\prec$ is referred to as* weak precedence *in [21].*

▶ **Definition 4** (Connect function [21], [39, Lemma 1]). *Given anchors $a \prec a'$, we define the following quantities (some shown in Figure 1):*

$$\begin{aligned}
\Delta_T(a, a') &:= t'_s - t_e \quad \text{and} \quad \Delta_Q(a, a') := q'_s - q_e && T\text{-gap and } Q\text{-gap} \\
\mathrm{L}_\infty(a, a') &:= \max\left(0, \Delta_T(a, a'), \Delta_Q(a, a')\right) && \text{max-gap} \\
\mathsf{ov}_T(a, a') &:= t_e - t'_s \quad \text{and} \quad \mathsf{ov}_Q(a, a') := q_e - q'_s && T\text{-overlap and } Q\text{-overlap} \\
\Delta_{\mathrm{diag}}(a, a') &:= |\Delta_T(a, a') - \Delta_Q(a, a')| = |\operatorname{diag}(a) - \operatorname{diag}(a')| && \text{diagonal distance}
\end{aligned}$$

*where $\operatorname{diag}(a) = t_s - q_s$ is the diagonal of $a$. Then the cost of connecting two anchors is obtained as the maximum of the $\mathrm{L}_\infty$ gap cost and the $\Delta_{\mathrm{diag}}$ overlap cost:*

$$\mathsf{connect}(a, a') = \max\left(\mathrm{L}_\infty(a, a'), \Delta_{\mathrm{diag}}(a, a')\right) = \begin{cases} \mathrm{L}_\infty(a, a') & \text{if } (t_e, q_e) \preceq (t'_s, q'_s) \text{ (gap-gap),} \\ \Delta_{\mathrm{diag}}(a, a') & \text{otherwise (overlap case).} \end{cases}$$

▶ **Definition 5** (Global and semi-global cost [21]). *Let $A = a_1, \ldots, a_c$ be a $\prec$-chain of anchors between $T$ and $Q$. We define the* global (alignment) cost *of $A$ as $\mathsf{gcost}(A) = \sum_{i=0}^{c} \mathsf{connect}(a_i, a_{i+1})$, where dummy anchors $a_0 = a_{\mathrm{start}} = ([0, 0), [0, 0))$ and $a_{c+1} = a_{\mathrm{end}} = ([|T|, |T|), [|Q|, |Q|))$ indicate the start and end of the alignment. The* semi-global cost $\mathsf{sgcost}(A)$ *ignores the initial and final gap in $T$, that is,*

$$\mathsf{sgcost}(A) = \Delta_Q(a_{\mathrm{start}}, a_1) + \left(\sum_{i=1}^{c-1} \mathsf{connect}(a_i, a_{i+1})\right) + \Delta_Q(a_c, a_{\mathrm{end}}).$$

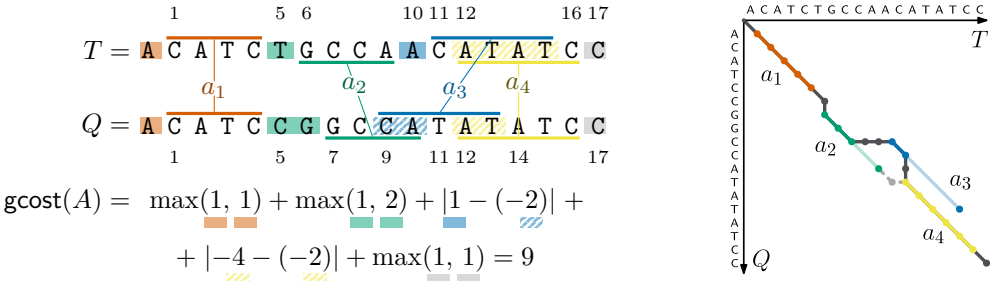



**Figure 2** Example of a chain $A = a_1, a_2, a_3, a_4$ of cost $\mathsf{gcost}(A) = 9$ (see Definitions 4 and 5, where on the left the gaps and overlaps between consecutive anchors are marked with solid and striped background, respectively. On the right, a possible alignment in the corresponding edit graph is shown that uses a prefix of each anchor. Skipping $a_3$ would reduce the cost from 9 to 6.

▶ **Problem 6** (Colinear chaining [21]). *Given a set $\mathcal{A}$ of $n$ exact match anchors, the* colinear chaining problem with $L_\infty$ gap costs and $\Delta_{\text{diag}}$ overlap costs *consists of finding an ordered subset $A = a_1, \ldots, a_c$ of $\mathcal{A}$ that is a $\prec$-chain and has minimum global cost* $\mathsf{gcost}(A)$. *The* semi-global *variant of the problem minimizes* $\mathsf{sgcost}(A)$ *instead.*

## 3.2 Variants, algorithms, and anchored edit distance

Jain, Gibney, and Thankachan [21] introduced and studied efficient solutions to Problem 6, as well as variants using different precedence formulations, which we will briefly discuss now. While the definition of non-overlapping anchors has been consistent in the literature, overlaps can be handled in multiple ways: for example, $([1..3), [1..3)), ([2..3), [5..6))$ is a $\prec$-chain but it aligns $T[2..3)$ to two different regions of $Q$, raising doubts about the usefulness of allowing this case. Luckily, the colinear chaining problems under $\prec$ and under the following formulations are equivalent,[2] and in turn they all compute the anchored edit distance.

▶ **Definition 7** (Alternative precedence formulations [21, 39]). *Anchor $a = ([t_s..t_e), [q_s..q_e))$* strictly precedes *$a' = ([t'_s..t'_e), [q'_s..q'_e))$, in symbols $a \prec_{\text{s}} a'$, if $a \neq a'$ and $t_s \leq t'_s$, $t_e \leq t'_e$, $q_s \leq q'_s$, $q_e \leq q'_e$. Also, we say that $a$* strongly precedes *$a'$, in symbols $a \prec_{\texttt{ChainX}} a'$, if $t_s < t'_s$, $t_e < t'_e$, $q_s < q'_s$, and $q_e < q'_e$.*

▶ **Theorem 8** ([21, Theorem 2] and [40, Theorem 3]). *For a fixed set of anchors $\mathcal{A}$, the following quantities are equal: the anchored edit distance (Definition 1) and the optimal $\prec$-chain, $\prec_{\text{s}}$-chain, and $\prec_{\texttt{ChainX}}$-chain global cost (*$\mathsf{gcost}$*, Definition 5). Similarly, the semi-global variant of anchored edit distance (i.e. initial and final $T$-gaps cost 0) is equal to the minimum semi-global cost* $\mathsf{sgcost}(A)$ *of an optimal $\prec$-, $\prec_{\text{s}}$-, or $\prec_{\texttt{ChainX}}$-chain $A$.*

There are already several dynamic-programming solutions to Problem 6 and its variants. Note how each precedence formulation (Definitions 3 and 7) defines a partial order on the input anchor set $\mathcal{A}$: consistently with the literature [46], the chaining algorithms consider $\mathcal{A}$ to be in a topological order. All solutions, including ours, compute values $C[j]$, the optimal partial costs $\mathsf{pgcost}(A) = \sum_{i=0}^{c-1} \mathsf{connect}(a_i, a_{i+1})$ over all chains $A = (a_0, a_1, \ldots, a_c)$ starting in $a_0 = a_{\text{start}}$ and ending in anchor $j$ (and thus not yet connecting to the final anchor $a_{\text{end}}$).

[2] We choose then to concentrate on $\prec$ and not one of its variants because the algorithmic results will be easier to prove under this precedence (see Lemma 13 and Baker and Giancarlo [3]).

Due to the additivity of the gcost function, values $C[j]$ can be computed in $O(n^2)$ time via dynamic programming (DP) as $C[0] = 0$ and

$$C[j] = \min_{i \in [0..j): a_i \prec a_j} C[i] + \mathsf{connect}(a_i, a_j) \qquad \forall j \in [1..n+1], \tag{1}$$

and it is easy to see that $C[n+1]$ is the minimum gcost of a $\prec$-chain. By changing how $a_0$ connects to all other anchors, a variant of Equation (1) also solves semi-global chaining.

Jain et al. also introduced sub-quadratic time solutions and a practical heuristic.

▶ **Theorem 9** ([21, Theorem 1]). *The colinear chaining problem with* $L_\infty$ *gap costs and* $\Delta_{\text{diag}}$ *overlap costs (Problem 6) can be solved in time:*

- $O(n \log^2 n)$ *for* $\prec_{\text{s}}$*-chains with fixed-length anchors;*
- $O(n \log^3 n)$ *for* $\prec$*-chains; and*
- $O(n \log^4 n)$ *for* $\prec_{\text{s}}$*-chains (which are now known to be equivalent to* $\prec$*-chains).*

Values $C[i]$ can be computed using dynamic multi-dimensional range minimum query data structures (a case of orthogonal range search in computational geometry [11]), but this was not implemented. Instead, [21] implements a practical algorithm that *approximates* the optimum $\prec_{\texttt{ChainX}}$-chain cost in $O(\text{SOL} \cdot n + n \log n)$ expected time, where SOL is the cost of the output chain and we assume a uniform distribution of start positions of the anchors in $Q$ or $T$ [21, Algorithm 2]. This solution, called `ChainX`, was tested on sequences of bacterial-like size and features: experimentally, `ChainX`'s output on maximal unique match (MUM) anchors correlates with edit distance and is faster than programs directly computing the edit distance, but the performance degrades significantly when the number of anchors increases [21, Section 6]. The approximate method `ChainX` was completed by Rizzo et al. into an exact algorithm `ChainX-opt` [39, Algorithm 1], shown to find a better score for a small number of cases with minimal practical slowdown compared to `ChainX`.

▶ **Theorem 10** ([39, Theorem 2] and [40, Corollary 2]). *The colinear chaining problems with* $L_\infty$ *gap costs and* $\Delta_{\text{diag}}$ *overlap costs under* $\prec$, $\prec_{\text{w}}$, *and* $\prec_{\texttt{ChainX}}$ *precedence can be solved in time* $O(n \log n + \text{OPT} \cdot n)$ *on average, where* OPT *is the optimum chain cost, assuming that* $n \leq \min\{|Q|, |T|\}$ *and the anchors are uniformly distributed in* $Q$*.*

**Properties of anchored edit distance.** We now list a few additional properties of the anchored edit distance. As implied by the name, the anchored edit distance is a variant on the classical edit distance, and admits a corresponding *edit graph* [28] that contains edges corresponding to insertions, deletions, and substitutions (all cost 1) and matches as indicated by the anchors (of cost 0), see Figure 2. We define $d_{ij}$ as the distance (length of the shortest path) from $(0, 0)$ to $(i, j)$. The proof of Theorem 8 gives the following fact.

▶ **Corollary 11.** *Let* $\mathcal{A} = a_1, \ldots, a_n$ *be sorted according to any* $\prec$*-topological order. The optimal partial cost* $C[j]$ *to the start of an anchor* $a_j$ *then equals* $d_{\mathsf{start}(a_j)}$*, the smallest distance from* $(0, 0)$ *to* $\mathsf{start}(a_j)$ *in the edit graph defined by* $\mathcal{A}$*.*

From this correspondence, it is clear [39, Corollary 1] that overlapping or touching anchors on the same diagonal can be merged, and similarly, that an anchor of length $\ell$ can be split into $\ell$ anchors of length 1, or into two anchors of lengths $\ell_1 + \ell_2 = \ell$, while keeping all partial costs the same.

### 3.3 Dynamic predecessor data structures

A data structure for dynamic predecessor queries supports the following operations on a set $S$ of totally ordered keys:

- insert(x): Add $x$ to $S$.
- remove(x): Remove $x$ from $S$.
- succ(x): Return $\min\{y \in S \mid y > x\}$, the smallest element larger than $x$.
- pred(x): Return $\max\{y \in S \mid y < x\}$, the largest element less than $x$.

In practice, keys can also have associated values, in which case get(x) can return this value in case $x \in S$.

Common implementations use balanced binary trees (red-black trees [17]) or B-trees [4], taking $O(n)$ words of space and $O(\log n)$ time for a set of $n$ elements. When the keys are integers up to $U$, Van Emde Boas trees [44] use $O(U)$ space and allow all operations in $O(\log \log U)$ time (in the word RAM model). Y-fast tries [47] improve this to $O(\log \log U)$ time in $O(n)$ space. Johnson's priority queue [22] is another variant often used in chaining methods that uses $O(U^{1/p})$ space for $O(p \log \log D)$ queries, where $D$ is the distance between the two elements surrounding the queried element $x$ and $p$ is a small positive integer parameter that can be chosen freely.

**With known keys.** In case the set of all $O(n)$ inserted keys $x$ is known in advance, they can be sorted in $O(n \log \log n)$ time and $O(n)$ space [19]. Then, each key can be assigned its *rank* from 1 to $n$ in the sorted list (e.g. via a lookup table), and the predecessor structure can use the ranks instead. Thus, this case allows for $O(\log \log n)$ rather than $O(\log \log U)$ queries while using $O(n)$ space.

**Incremental suffix minimum queries.** We can also use predecessor structures for *incremental suffix minimum queries*, where we insert keys $x$ with an associated value $f(x)$ (but never remove them) and get queries that ask for $\mathsf{smq}(x) = \min\{f(y) \mid y \in S : y \geq x\}$. A standard way to support this is the following: if we try to insert $(x, f(x))$ but $f(x) \geq \mathsf{smq}(x)$, do nothing; otherwise, insert $(x, f(x))$ (or update the value associated with $x$), and repeatedly find $y = \mathsf{pred}(x)$, then remove $y$ as long as $f(x) \leq f(y)$. If $S$ is known in advance, insertions and smq queries can both be supported in $O(\log \log n)$ amortized time.

## 4 Chaining in $O(n \log \log n)$ time

We now give an algorithm for colinear chaining with $\mathrm{L}_\infty$ gap costs and $\Delta_{\mathrm{diag}}$ overlap costs on the input set $\mathcal{A} = \{a_1, \ldots, a_n\}$ (Problem 6) in $O(n \log \log n)$ time and $O(n)$ space. We assume that $\mathcal{A}$ is sorted in lexicographical order of 2D points $\mathsf{start}(a)$. We do not worry about ties, because we also assume that $\mathcal{A}$ does not contain perfectly-overlapping or touching anchors (on the same diagonal), as these can be merged into longer anchors without changing the anchor-supported character matches and thus the optimal gcost chain (Theorem 8). These two assumptions can be enforced in $O(n \log \log n)$ time and $O(n)$ space by sorting the anchors first by diagonal and then by anti-diagonal using [19] and then merging adjacent overlapping anchors.

We follow the same general strategy as all sub-quadratic-time chaining algorithms (see e.g. [14, 3, 21]): we partition the minimum of Equation (1) into four disjoint cases described by the connect function and use a line sweep of the 2D plane $[0..|T|) \times [0..|Q|)$. Consider Figure 3 and the following equation:

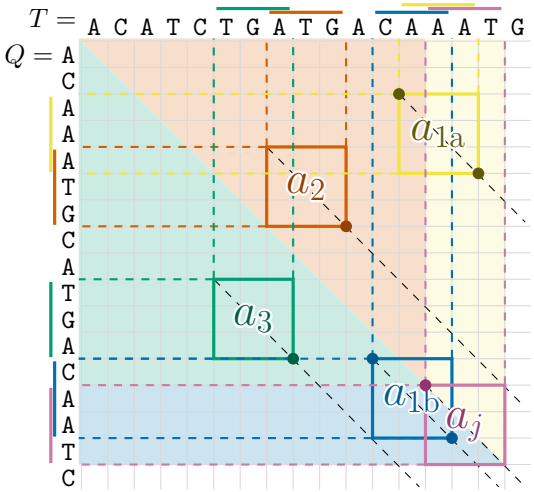



**Figure 3** Anchors $a_{1a}$, $a_{1b}$ $a_2$, $a_3$ showing the four mutually exclusive cases for the computation of $C[j]$ (Equation (2)): overlap case with bigger $T$-overlap (1a) or bigger $Q$-overlap (1b), gap-gap with bigger $Q$-gap (2), and gap-gap with bigger $T$-gap (3). The areas containing the endpoint for each recursive anchor $a_i$ are highlighted with different colors, and the startpoints of $a_{1a}$, $a_{1b}$ (● and ● markers) are also highlighted to indicate that in Equations (3) and (4) they need to be checked.

$$C[j] = \min\left(C^{1a}[j], C^{1b}[j], C^2[j], C^3[j]\right), \qquad \text{where} \tag{2}$$

$$C^{1a}[j] := \min_{\substack{i\in[0..j):\\ a_i \prec a_j,\ \mathsf{ov}_T(a_i,a_j)>0,\\ \mathrm{diag}(a_i)\geq\mathrm{diag}(a_j)}} C[i] + \mathrm{diag}(a_i) - \mathrm{diag}(a_j) \quad \text{(overlap, larger } T\text{-overlap)} \tag{3}$$

$$C^{1b}[j] := \min_{\substack{i\in[0..j):\\ a_i \prec a_j,\ \mathsf{ov}_Q(a_i,a_j)>0,\\ \mathrm{diag}(a_i)<\mathrm{diag}(a_j)}} C[i] + \mathrm{diag}(a_j) - \mathrm{diag}(a_i) \quad \text{(overlap, larger } Q\text{-overlap)} \tag{4}$$

$$C^2[j] := \min_{\substack{i\in[0..j):\\ \mathsf{end}(a_i)\preceq\mathsf{start}(a_j),\\ \mathrm{diag}(a_i)>\mathrm{diag}(a_j)}} C[i] + \Delta_Q(a_i, a_j) \quad \text{(gap-gap, larger } Q\text{-gap)} \tag{5}$$

$$C^3[j] := \min_{\substack{i\in[0..j):\\ \mathsf{end}(a_i)\preceq\mathsf{start}(a_j),\\ \mathrm{diag}(a_i)\leq\mathrm{diag}(a_j)}} C[i] + \Delta_T(a_i, a_j) \quad \text{(gap-gap, larger } T\text{-gap)} \tag{6}$$

where $\min(\emptyset) = +\infty$. Then, our algorithm can be divided into three main stages, as partially described in Algorithm 1:

**Pre-processing.** First, we compute the sorted orders for the "left-to-right" (i.e. $t_s$ and $t_e$) and "top-to-bottom" (i.e. $q_s$ and $q_e$) line sweeps of the set of anchor startpoints and endpoints, and we additionally sort the anchors by their diagonal value $\mathrm{diag}(a)$ to compute $\mathsf{rank}_{\mathrm{diag}(a)}$, where $\mathsf{rank}_{\mathrm{diag}(a)} = d$ if $\mathrm{diag}(a)$ is the $d$-th hit diagonal. As explained in Section 3.3, this can be done in $O(n \log\log n)$ time and $O(n)$ space, since sorting considers at most $2n$ points. Following Baker and Giancarlo [3], for each $j \in [1..n]$ we compute anchors $\hat{a}_j$ and $\overleftarrow{a_j}$, respectively the closest anchor overlapping $\mathsf{start}(a_j)$ in a bigger diagonal (i.e. "above") and a smaller diagonal (i.e. "left"), with two line sweeps as detailed in Section 4.1 ($\hat{a}_j$ and $\overleftarrow{a_j}$ are called $visl(a_j)$ and $visa(a_j)$ in [3]). This will allow handling both overlap cases in constant time during the main line sweep.

**Algorithm 1** High-level description of chaining with $L_\infty$ gap costs and $\Delta_{\text{diag}}$ overlap costs.

**Input :** Anchors $\mathcal{A} = \{a_1, \ldots, a_n\}$ between $T$ and $Q$ such that no two anchors on the same diagonal overlap or have an empty gap (i.e. $\Delta_T > 0$ or $\Delta_Q > 0$)
**Output :** The minimum global chaining cost (gcost) over all $\prec$-chains

```
Sort anchors a ∈ A by diagonal value diag(a), to compute D := {diag(a) : a ∈ A};
preprocess_1a(A) ;                                   ▷ Computes anchors â
preprocess_1b(A);                                    ▷ Computes anchors ←a
Sort startpoints and endpoints
  P = {(a_j.start, j) : a_j ∈ A} ∪ {(a_j.end, j) : a_j ∈ A} ∪ {((0,0),0), ((|T|,|Q|), n+1)}
  in increasing T-coordinate order (in breaking ties, endpoints are smaller than
  startpoints and ≺ is used for points of the same type);
init_case_2(A);
init_case_3(A);
C[0] ← 0;
for (p, j) ∈ P in sorted order do
    if p = a_j.start then
        C^1a[j] ← C[â_j] + Δ_diag(â_j, a_j);          ▷ Lemma 13
        C^1b[j] ← C[←a_j] + Δ_diag(←a_j, a_j);        ▷ Lemma 13
        C^2[j] ← process_startpoint_case_2(a_j);
        C^3[j] ← process_startpoint_case_3(a_j);
        C[j] = min(C^1a[j], C^1b[j], C^2[j], C^3[j]); ▷ Equation (2)
    else                                              ▷ p = a_j.end
        process_endpoint_case_2(a_j);
        process_endpoint_case_3(a_j);
return C[n+1];          ▷ Best pgcost to final dummy anchor a_{n+1} = a_end
```

**Main line sweep.** We compute values $C[j]$ while performing a line sweep of the anchor startpoints and endpoints, from smaller to larger $T$-coordinate value. Following Eppstein et al. [14] (see also [9, 3]), when a startpoint $\mathsf{start}(a_j)$ is processed, $C[j]$ is calculated as per Equation (2) using $\hat{a}_j$, $\overleftarrow{a}_j$, and the data structures for the two gap-gap cases 2 and 3 as we will describe in Sections 4.2 and 4.3. When a startpoint or endpoint is reached (endpoints are processed before startpoints if they fall on the same $T$-position), these gap-gap data structures are updated.

**Post-processing.** Value $C[n+1]$ is equal to the anchored edit distance. Instead of augmenting our solution to store backtracking information (see e.g. [39]), we can optionally recover an optimal $\prec$-chain from values $C[j]$ in $O(n)$ time: we start from $j = n+1$ and we iteratively perform a linear search to find a predecessor $i < j$ such that $C[i] + \mathsf{connect}(a_i, a_j) = C[j]$ and we set $j = i$, until $j = 0$.

The rest of this section is dedicated to showing how to solve the recursive cases individually, in order of hardness: the overlap cases (1a and 1b), the bigger $Q$-gap case (2), and the bigger $T$-gap case (3). The correctness will follow from Equation (2) by induction on $j = 0, \ldots, n+1$, obtaining our main theoretical result thanks to Theorem 8.

▶ **Theorem 12.** *The anchored edit distance (Definition 1) can be computed in $O(n \log \log n)$ time and $O(n)$ space, by solving the colinear chaining problem with $L_\infty$ gap costs and $\Delta_{\text{diag}}$ overlap costs (Problem 6) in the same time and space.*

## 4.1 Overlap cases 1a and 1b

We study the computation of values $C^{1a}[j]$ (Equation (3)), as case $C^{1b}$ is solved in a symmetrical way. We are in this case when $a_i \prec a_j$, $\mathsf{ov}_T(a_i, a_j) > 0$, and $\mathrm{diag}(a_i) \geq \mathrm{diag}(a_j)$, meaning that the anchors from $C^{1a}[j]$ are exactly anchors $a_i$ such that: $(i)$ their $T$-interval $[t^i_s..t^i_e)$ contains $t^j_s$; and $(ii)$ $\mathrm{diag}(a_i) \geq \mathrm{diag}(a_j)$. Note that conditions $i$ and $ii$ are sufficient for the colinearity $a_i \prec a_j$. By adapting an argument of Baker and Giancarlo [3, Section 3.1], we can prove that we only need to check the closest anchor above $a_j$, as all other anchors cannot be a better case-1a predecessor.

▶ **Lemma 13.** *In the overlap case with bigger $T$-overlap (Equation (3)) and under the assumption that no two anchors overlap on the same diagonal, the optimal partial cost $C^{1a}[j]$ to anchor $a_j$ is equal to $C[\hat{a}] + \Delta_{\mathrm{diag}}(\hat{a}, a_j)$, where $\hat{a}$ is the anchor with the smallest $\mathrm{diag}(\hat{a})$ bigger than $\mathrm{diag}(a_j)$ such that $t^j_s \in [\hat{t_s}..\hat{t_e})$. If $\hat{a}$ is not defined, then $C^{1a}[j] = +\infty$.*

**Proof.** Assume by contradiction that $a' = ([t'_s..t'_e), [q'_s..q'_e))$ is a strictly better overlapping anchor considered by case 1a which is farther away, that is, $t^j_s \in [t'_s..t'_e)$, $\mathrm{diag}(a_j) \leq \mathrm{diag}(\hat{a}) < \mathrm{diag}(a')$ (no two anchors overlap in the same diagonal by assumption), and $C[a'] + \mathrm{diag}(a') - \mathrm{diag}(a_j) < C[\hat{a}] + \mathrm{diag}(\hat{a}) - \mathrm{diag}(a_j)$. If $a' \prec \hat{a}$, we reach a contradiction, as $C[a'] + \mathsf{connect}(a', \hat{a}) < C[\hat{a}]$ (case 1a) violates the definition of $C[\hat{a}]$ (Equation (1)). Otherwise $t'_s \in (\hat{t_s}..\hat{t_e})$ and we can split anchor $\hat{a}$ into two anchors $\hat{a}^1$ and $\hat{a}^2$, the first covering $[\hat{t_s}..t'_s)$ and the second covering $[t'_s..\hat{t_e})$. Due to Corollary 11, the partial optimal costs $C'$ of the modified instance $(\mathcal{A} \setminus \{\hat{a}\}) \cup \{\hat{a}^1, \hat{a}^2\}$ are identical for all anchors but $C'[\hat{a}^1] = C'[\hat{a}^2] = C[\hat{a}]$. But then $a' \prec \hat{a}^2$ by construction and $C'[a'] + \mathsf{connect}(a', \hat{a}^2) < C'[\hat{a}^2]$, reaching a contradiction. ◀

Then, as in [3, Section 3.3], we can compute anchors $\hat{a}_j$ with a $T$-coordinate sweep line and a successor data structure on the set $S \subseteq \mathcal{D} := \{\mathrm{diag}(a) : a \in \mathcal{A}\}$ of active anchors overlapping with the sweep line, as implemented in Algorithm 2. Since $|\mathcal{D}| \leq n$ and $\mathcal{D}$ can be precomputed, the computation takes $O(n \log\log n)$ time and $O(n)$ space. The overlap case 1b is symmetrical and can be obtained by sorting the points by $Q$-coordinate and using predecessor queries (see line 8 of Algorithm 2).

## 4.2 Gap-gap case, bigger gap in $Q$

If we are in this case, that is, if $\mathsf{end}(a_i) \preceq \mathsf{start}(a_j)$ and $\mathrm{diag}(a_i) > \mathrm{diag}(a_j)$, then $a_i \prec a_j$ and

$$C^2[j] = \min_{\substack{i \in [0..j): \\ \mathsf{end}(a_i) \preceq \mathsf{start}(a_j), \\ \mathrm{diag}(a_i) > \mathrm{diag}(a_j)}} \big(C[i] + q^j_s - q^i_e\big) = q^j_s + \min_{\substack{i \in [0..j): \\ \mathsf{end}(a_i) \preceq \mathsf{start}(a_j), \\ \mathrm{diag}(a_i) > \mathrm{diag}(a_j)}} \big(C[i] - q^i_e\big). \tag{7}$$

In geometric terms, we need to consider each anchor whose endpoint is in the first octant from $\mathsf{start}(a_j)$, as shown in Figure 3. This octant is characterized by the two equations $t^i_e \leq t^j_s$ $(i)$ and $\mathrm{diag}(a_i) > \mathrm{diag}(a_j)$ $(ii)$: the former is guaranteed by processing the points from left to right, and the diagonal constraint is handled via a suffix minimum query over diagonal keys, as shown in Algorithm 3. Indeed, from $(i)$ and $(ii)$ we can derive that $q^j_e \leq q^i_s$ and thus $\mathsf{end}(a_i) \preceq \mathsf{start}(a_j)$ as follows:

$$\begin{aligned} \mathrm{diag}(a_i) > \mathrm{diag}(a_j) &\iff t^i_e - q^i_e > t^j_s - q^j_s \\ &\iff q^j_s - q^i_e > t^j_s - t^i_e \geq 0 && (t^i_e \leq t^j_s) \\ &\implies q^j_e \leq q^i_s. \end{aligned}$$

**Algorithm 2** Procedure handling the computation of anchors $\hat{a}_j$ for $j \in [1..n]$, where $\hat{a}_j$ is defined as the anchor overlapping with $a_j$ with the smallest diagonal greater than $\text{diag}(a_j)$, if it exists, otherwise $\hat{a} = \perp$, the dummy anchor. Note that we assume anchor set $\mathcal{A}$ to not have any anchors overlapping or touching along the same diagonal, so no two startpoints or endpoints can be equal.

```
Procedure preprocess_1a(A)
    Sort startpoints and endpoints P = {(a.start, a) : a ∈ A} ∪ {(a.end, a) : a ∈ A} in
      increasing T-coordinate order (in breaking ties, endpoints are smaller);
    â_0 ← ⊥;
    â_{n+1} ← ⊥;
    S ← ∅;                    ▷ predecessor data structure over D = {diag(a) : a ∈ A}
    for (p, a) ∈ P in sorted order do
        if p = a.start then
            â ← S.succ(diag(a)).value ;                    ▷ if it exists, otherwise ⊥
            S.insert(diag(a) ↦ a);
        else                                                ▷ p = a.end
            S.remove(diag(a));
```

**Algorithm 3** Procedures used in Algorithm 1 to handle the computation of $C^2[j]$, the optimal partial cost $\mathsf{pgcost}(A)$ of any $\prec$-chain $A$ ending at anchor $a_j$ with a predecessor $a_i$ in the gap-gap, bigger $Q$-gap case (see Figure 3 and Equation (2)).

```
Procedure init_case_2(A)
    S ← ∅;                    ▷ suffix min query structure over D = {diag(a) : a ∈ A}
    S.insert(diag(a_0) ↦ 0);                          ▷ base case for a_0 = a_start
Procedure process_startpoint_case_2(a_j)
    return q_s^j + S.smq(diag(a_j) + 1);
Procedure process_endpoint_case_2(a_i)
    S.insert(diag(a_i) ↦ C[i] − q_e^i);          ▷ insert or update the recursive value
```

We briefly discuss Algorithm 3. As originally presented in [14, Sections 2 and 4] for similar recursions, the recursive values $C[i] - q_{\text{e}}^i$ are stored in a one-dimensional data structure, as our line sweep conveniently gives us one half-plane for free. Since the set of $\text{diag}(a)$ keys is known in advance and the number of startpoints and endpoints is $2n$, we can handle case 2 in $O(n \log \log n)$ time and $O(n)$ space.

## 4.3 Gap-gap case, bigger gap in $T$

This is the final and hardest case to handle for the computation of Equation (2). Even though it is symmetrical to case 2, we do not get a half-plane filter for free as our sweeping of the plane goes from left to right (see Figure 3) while the constraint is on $q$, and this cannot be easily handled with a one-dimensional data structure or by changing the sweep line. Indeed, in recent chaining results this type of recursive case has been handled with 2D search data structures, resulting in final time complexities of $O(n \log^2 n)$ or $O(n \log n \log \log n)$ [34, 37, 21]. Instead, using the space-partitioning technique of Eppstein et al. [14], we can efficiently represent the optimal recursive cases projected onto the sweep line as dynamic

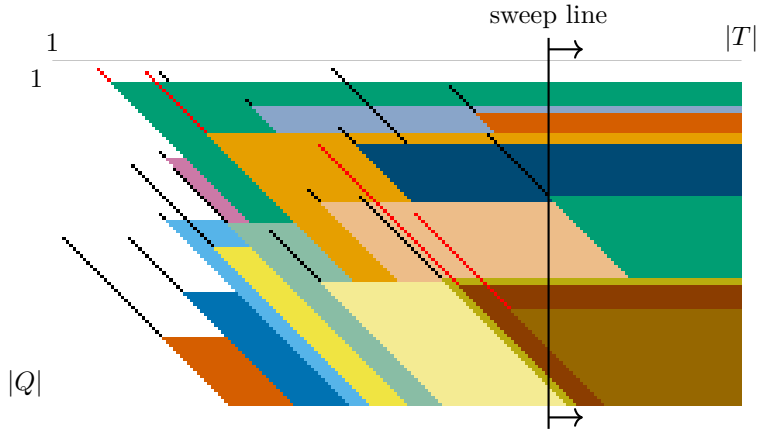



**Figure 4** Optimal recursive values for case 3 (Equation (2)) handled in the main left-to-right sweep line. The anchors of an optimal $\prec$-chain are shown in red, whereas the other anchors of $\mathcal{A}$ are black. Each color corresponds to a region "owned" by a different anchor $a_i$, so values $R_t[q]$ of the owner array (Equation (9)) can be deduced by the color at the sweep line.

one-dimensional segments, as described in the rest of this section. Our goal is to present a sufficiently detailed variant of the $O(n \log \log n)$-time technique that runs in $O(n)$ space, which is not a widely known result and might be of independent interest.[3]

Symmetrically to Equation (7), we can rewrite $C^3[j]$ as

$$C^3[j] = t^j_{\mathrm{s}} + \min_{\substack{i \in [0..j): \\ \mathsf{end}(a_i) \preceq \mathsf{start}(a_j), \\ \mathrm{diag}(a_i) \leq \mathrm{diag}(a_j)}} \big(C[i] - t^i_{\mathrm{e}}\big). \tag{8}$$

From this we can derive the following first main insight: each anchor $a_i$ defines a *forward region of influence* where it can precede a following anchor $a_j$ in the sense of Equation (8). This region (also called "area of influence") is triangular and delimited by the horizontal and diagonal lines starting from endpoint $\mathsf{end}(a_i)$ (see Figure 4): for any successor $a_j$ with $\mathsf{start}(a_j) = (t, q)$, we consider the intersection of the half-plane $q^i_{\mathrm{e}} \leq q$ bounded by the horizontal line through $\mathsf{end}(a_i)$ and the half-plane $\mathrm{diag}(t^j_{\mathrm{s}}, q^j_{\mathrm{s}}) \geq \mathrm{diag}(a_i)$ bounded by the diagonal line through $\mathsf{end}(a_i)$. When the areas of multiple anchors intersect, those with the smaller value $C[i] - t^i_{\mathrm{e}}$ take precedence, as remarked by the following observation.

▶ **Observation 14** (Chao and Miller [9, Lemma 1]). *Consider any two anchors* $a_i = ([t^i_s..t^i_e), [q^i_s..q^i_e))$ *and* $a_{i'} = ([t^{i'}_s..t^{i'}_e), [q^{i'}_s..q^{i'}_e))$. *For all anchors* $a_j$ *such that* $a_j$ *is case-*3 *successor of both* $a_i$ *and* $a_{i'}$ *(as in Equation* (8)*), in symbols* $\mathsf{end}(a_i) \preceq \mathsf{start}(a_j)$, $\mathsf{end}(a_{i'}) \preceq \mathsf{start}(a_j)$, $\mathrm{diag}(a_i) \leq \mathrm{diag}(a_j)$, *and* $\mathrm{diag}(a_{i'}) \leq \mathrm{diag}(a_j)$, *we have that the value* $\min(C[i] - t^i_e, C[i'] - t^{i'}_e)$ *does not depend on the exact position of* $a_j$.

Let $a_j = ([t^j_{\mathrm{s}}..t^j_{\mathrm{e}}), [q^j_{\mathrm{s}}..q^j_{\mathrm{e}}))$ be an arbitrary anchor starting at the current sweep-line coordinate $t$, that is, $t^j_{\mathrm{s}} = t$, and consider Figure 4. We now define how the optimal recursive anchors for $a_j$ change based on where the startpoint of $a_j$ is along the sweep line (i.e. its $Q$-coordinate $q^j_{\mathrm{s}}$) as follows, with ties in the arg min broken to larger $i$:

[3] The original efficient chaining solutions by Eppstein et al. [14] and Chao and Miller [9] used $O(|T| + |Q|)$ space. Baker and Giancarlo do claim to use $O(n)$ space [3, Theorem 4.1] but do not provide an explicit space-complexity proof for handling the intersection points [3, p. 251] as we will later do in Lemma 16. Other methods use $O(n)$ space but require $O(n \log \log(|T| + |Q|))$ time.

$$R_t[q] = \operatorname*{arg\,min}_{\substack{i\in[0..n):\\ t^i_{\mathrm{e}}\le t,\, \mathrm{diag}(a_i)\le t-q}} \big(C[i]-t^i_{\mathrm{e}}\big), \qquad \text{where } \arg\min(\emptyset)=\perp \quad \text{and} \quad q\in[0..|Q|]. \tag{9}$$

We call $R_t$ the *owner array*, because value $R_t[q]$ is equal to the anchor index the area belongs to: by construction we have $C^3[j] = t^j_{\mathrm{s}} + C[i] - t^i_{\mathrm{e}}$ with $i = R_{t^j_{\mathrm{s}}}[q^j_{\mathrm{s}}]$. Note that when there is no anchor $a_i$ such that $\mathsf{end}(a_i) \preceq (t,q)$, then $R_t[q] = \perp$, where $\perp$ is the dummy anchor, and $C^3[j] = +\infty$.

The owner array has size $O(|Q|)$, so it would be too expensive to maintain it as a flat array. Instead, Eppstein et al. [14] employ a compact $O(n)$-space version of $R_t$ as a doubly-linked list (called OWNER list) using the following second main insight: while the areas of influence can have complex interactions and shapes as seen in Figure 4, they are always delimited by some horizontal or diagonal line starting in an anchor endpoint.

▶ **Observation 15** ([14]). *For all changes in anchor index $R_t[q] \neq R_t[q+1]$ with $q \in [1..|Q|]$, there is a unique* active dividing line *such that the following holds for some unique anchor $a_i$, $i \in [0..n]$:*

- *(horizontal dividing line) the active line is $q+1 = q^i_e$ and $R_t[q+1] = i$, that is, $a_i$ owns the region of influence* starting *at $q+1$; or*
- *(diagonal dividing line) the active line is* $\mathrm{diag}(t,q) = \mathrm{diag}(a_i)$ *and $R_t[q] = i$, that is, $a_i$ owns the area of influence* ending *at $q$.*

Then, as shown in Algorithm 4, we can represent $R_t$ in a compressed $O(n)$-space representation consisting of two predecessor data structures of diagonals mapped to anchors and a doubly-linked list of active lines from Observation 15. These lines, which are separately indexable via two dynamic predecessor data structures (Section 3.3), are used to partition the sweep line regions with different $R_t[q]$ values. Note that Observation 15 does not indicate which anchor owns the region between a diagonal and a horizontal line, hence we need to also store the different values of $R_t[q]$ as associated values of the elements in the list. We obtain the following result and thus complete the proof of Theorem 12.

▶ **Lemma 16.** *We can compute values $C^3[j]$ (Equation (8)) as part of the main left-to-right line sweep (Algorithm 1) in $O(n \log\log n)$ time and $O(n)$ space.*

**Proof sketch.** The correctness follows from Equation (8), Observations 14 and 15, and from the correct handling of the compressed representation of $R_t$ as partially shown in Algorithm 4 and fully shown in [14, Section 4]. As discussed in [14], the total number of diagonal or horizontal lines is $2n$ and the cost of handling intersections can be charged to the diagonal and horizontal lines of origin. The only modification to the solution by Eppstein et al. is that to reach both $O(n \log\log n)$ time and $O(n)$ space, we delay the intersection updates until we reach the closest column to the right of it that contains an anchor start or end, and we execute these updates in an unsorted, first-in-first-out order. We argue that this modification maintains correctness, since in a batch of intersection updates we remove areas of influence without changing the order (i.e. we get a subsequence of $R_t$ and of the active dividing lines). We can check for new intersection points at any modification of $R_t$: if they occur before the current $t$ they are handled directly; otherwise they are queued accordingly in $O(\log\log n)$ time per point using a dynamic predecessor query structure on the $T$-positions hit by some anchor startpoint or endpoint. We leave the full pseudocode implementation and a more detailed proof for a future extended version of this work. ◀

**Algorithm 4** Partial implementation of the handling of the gap-gap, bigger $T$-gap case (Equation (2)), as part of Algorithm 1, and following Eppstein et al. [14]. The doubly-linked list $\overline{R}$ is a compact representation of the owner array $R_t$ (Equation (9)) of elements $(d, a) \in \{\rightarrow, \searrow\} \times \mathcal{A}$ mapping each dividing line to the originating anchor $a_{R_t[q]}$. We omit the handling of intersection updates, and we omit the edge cases in `process_endpoint_case_3` where $\mathsf{end}(a_i)$ falls on top of an active diagonal or horizontal line.

**Procedure** `init_case_3`$(\mathcal{A})$
  $H \leftarrow \emptyset$; ▷ **predecessor data structure over** $\mathcal{H} = \{q_{\mathrm{e}} : ([t_{\mathrm{s}}..t_{\mathrm{e}}), [q_{\mathrm{s}}..q_{\mathrm{e}})]) \in \mathcal{A}\}$
  $D \leftarrow \emptyset$; ▷ **predecessor data structure over** $\mathcal{D} = \{\mathrm{diag}(a) : a \in \mathcal{A}\}$
  $H.\mathsf{insert}(0 \mapsto a_0)$; ▷ **base case for** $a_0 = a_{\mathrm{start}}$
  $D.\mathsf{insert}(0 \mapsto a_0)$; ▷ **base case for** $a_0 = a_{\mathrm{start}}$
  $\overline{R} \leftarrow \big(((\rightarrow, a_0) \mapsto a_0), ((\searrow, a_0) \mapsto \perp)\big)$;
  ▷ **doubly-linked list of dividing lines (Observation 15) representing** $R_t$**, where each line is mapped to anchor** $a_{R_t[q]}$ **of the corresponding region.**
  $Q \leftarrow \big\{t \mapsto \emptyset : t \in \{t^j_{\mathrm{s}} : j \in [1..n]\} \cup \{t^j_{\mathrm{e}} : j \in [1..n]\}\big\}$;
  ▷ **Intersection update queues before each** $T$**-coordinate in the input.**

**Procedure** `process_startpoint_case_3`$(a_j)$
  Process all intersection updates in $Q[t^j_{\mathrm{s}}]$;
  $a_{\rightarrow} \leftarrow H.\mathsf{pred}(q^j_{\mathrm{s}} + 1).\mathsf{value}$; ▷ **closest** $\rightarrow$ **at or above** $q^j_{\mathrm{s}}$
  $a_{\searrow} \leftarrow D.\mathsf{succ}(\mathrm{diag}(a_j) - 1).\mathsf{value}$; ▷ **closest** $\searrow$ **at or above** $\mathrm{diag}(a_j)$
  **if** $q_s^{\rightarrow} < t^j_s - \mathrm{diag}(a_{\searrow})$ **then** ▷ $(\rightarrow, a_{\rightarrow})$ **is closer**
    $a_i \leftarrow \overline{R}.\mathsf{node}(\rightarrow, a_{\rightarrow}).\mathsf{value}$;
  **else** ▷ $(\searrow, a_{\searrow})$ **is closer**
    $a_i \leftarrow \overline{R}.\mathsf{node}(\searrow, a_{\searrow}).\mathsf{value}$;
  **return** $q^j_{\mathrm{s}} + C[i] - t^i_{\mathrm{e}}$; ▷ **Equation (8)**

**Procedure** `process_endpoint_case_3`$(a_i)$
  Process all intersection updates in $Q[t^i_{\mathrm{e}}]$;
  $a_{\rightarrow} \leftarrow H.\mathsf{pred}(q^i_{\mathrm{e}}).\mathsf{value}$; ▷ **closest** $\rightarrow$ **above** $q^i_{\mathrm{e}}$
  $a_{\searrow} \leftarrow D.\mathsf{succ}(\mathrm{diag}(a_i)).\mathsf{value}$; ▷ **closest** $\searrow$ **above** $\mathrm{diag}(a_i)$
  **if** $q_s^{\rightarrow} < t^i_e - \mathrm{diag}(a_{\searrow})$ **then** ▷ $(\rightarrow, a_{\rightarrow})$ **is closer**
    $a_k \leftarrow \overline{R}.\mathsf{node}(\rightarrow, a_{\rightarrow}).\mathsf{value}$;
    **if** $C[i] - t^i_e \le C[k] - t^k_e$ **then**
      $\overline{R}.\mathsf{node}(\rightarrow, a_{\rightarrow}).\mathsf{insert_next}((\rightarrow, a_i) \mapsto a_i)$;
      $\overline{R}.\mathsf{node}(\rightarrow, a_i).\mathsf{insert_next}((\searrow, a_i) \mapsto a_k)$;
  **else** ▷ $(\searrow, a_{\searrow})$ **is closer**
    $a_k \leftarrow \overline{R}.\mathsf{node}(\searrow, a_{\searrow}).\mathsf{value}$;
    **if** $C[i] - t^i_e \le C[k] - t^k_e$ **then**
      $\overline{R}.\mathsf{node}(\searrow, a_{\searrow}).\mathsf{insert_next}((\rightarrow, a_i) \mapsto a_i)$;
      $\overline{R}.\mathsf{node}(\rightarrow, a_i).\mathsf{insert_next}((\searrow, a_i) \mapsto a_k)$;
  $H.\mathsf{insert}(\mathrm{diag}(a_i) \mapsto a_i)$; ▷ **line** $(\rightarrow, a_i)$ **becomes active**
  $D.\mathsf{insert}(\mathrm{diag}(a_i) \mapsto a_i)$; ▷ **line** $(\searrow, a_i)$ **becomes active**
  If $\overline{R}.\mathsf{node}(\rightarrow, a_i).\mathsf{prev}$ is a diagonal line, queue their intersection update at $Q[t]$ with $t$ the closest $T$-coordinate after or including the intersection;
  If $\overline{R}.\mathsf{node}(\searrow, a_i).\mathsf{next}$ is a horizontal line, queue their intersection update at $Q[t]$ with $t$ the closest $T$-coordinate after or including the intersection;

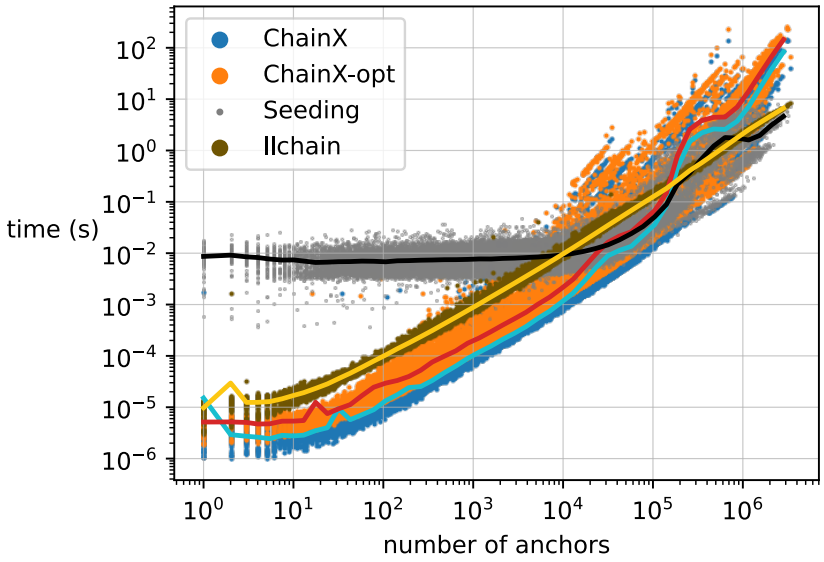



**Figure 5** Time spent on seeding (grey, searching MEMs using MUMmer4) and chaining the resulting anchors using `ChainX`, `ChainX-opt`, and `llchain` for MEMs of length $\geq 50$ between 100 000 HiFi reads and a human genome reference. The highlighted curves indicate the average running time after bucketing by the number of anchors.

## 5 Experiments

We implemented our algorithm in `llchain` and compare it to `ChainX` and `ChainX-opt`. Our implementation uses a standard C++ `std::map` as the predecessor structure and a segment tree for suffix minimum queries, with $O(\log n)$ time operations, and uses the `std::sort` and `std::stable_sort` sorting implementations. Experiments are run on an Intel Xeon Gold 6148 2.40GHz machine with 376 GB of RAM in the University of Helsinki Kale cluster (the Slurm job is run with options `--exclusive` and `--mem=60G`).

Similar to [39], we consider 100 000 PacBio HiFi reads of a human genome[4] as generated by the T2T project. We then use MUMmer4 [12, 23, 26] to find all maximal exact matches (MEMs) of length at least 50 between each read and the concatenation of all chromosomes in the T2T-CHM13 human genome reference [31, 38]. We exclude from the analysis 216 reads (0.2%) presenting erroneous inexact matches reported by MUMmer4.[5] For each method, we measure how long it takes to sort the MEMs, double check the absence of overlapping or touching anchors on same diagonal[6], and then find the best semi-global chain. We call each method directly via its API.

Results can be seen in Figure 5. `llchain` is around 10× slower than `ChainX` and 4× slower than `ChainX-opt` for 100 to 10 000 anchors, but in this case, the time needed for seeding dominates. When the number of anchors is large (above 100 000), `ChainX` and `ChainX-opt` have a high variance in their running time, and likely slow down when there

[4] Run m64004 available at `https://s3-us-west-2.amazonaws.com/human-pangenomics/index.html?prefix=T2T/scratch/HG002/sequencing/hifi/`.
[5] See `https://github.com/mummer4/mummer/issues/235`.
[6] Note that `llchain` can handle an arbitrary anchor set in input, since if the check fails it joins any anchor pair $a, a'$ such that $\mathsf{connect}(a, a') = 0$: the anchored edit distance is unaffected.

are many matches in repetitive regions. In these cases, the running time grows roughly quadratically, and the slow instances dominate the overall running time. On the other hand, `llchain` has a very consistent running time and, as predicted by the theory, has a complexity roughly linear in the number of anchors. Overall, seeding takes on average 0.044s, `ChainX` 0.133s, `ChainX-opt` 0.243s, `llchain` 0.056s. Thus, `llchain` is only slightly slower than the seeding on average, whereas the largest cases hurt the other methods, making them over 2.3× slower. The exact methods `ChainX-opt` and `llchain` always agree on the optimal cost, and consistently with the chaining of maximal unique matches [39] `ChainX` returns a suboptimal chain for 2904 reads (2.9% of the reads considered): for this subset of reads, the median improvement of the anchored edit distance is 1296, and the median improvement ratio is significant, 80×. Separate fuzz tests show that `llchain` gives the same results as a naive $O(n^2)$-time implementation of Equation (2).

## 6 Conclusion

Our new algorithm for anchored edit distance combines many existing ideas and runs in $O(n \log \log n)$ time and $O(n)$ space. Unlike the original chaining algorithm with this time complexity [14, 9], it achieves $O(n)$ space rather than e.g. $O(|Q| + |T|)$ space, which might be of independent interest. While a faithful $O(n \log \log n)$-time implementation is unlikely to be competitive in practice (see e.g. [30]), we have showed the $O(n \log n)$-time behavior of our simplified `llchain` in a realistic long-read setting. We further note that our implementation is not yet optimized, and in particular that `std::map` and `std::list` are known to be slow. Future versions of `llchain` will use B-trees or one of the engineered predecessor implementations of Dinklage et al. [13], which are reported to be respectively 4× and 16× faster than `std::map`, and it should be possible to remove the linked list, as it is implicitly represented by the predecessor structures on horizontal and diagonal lines.

Another current limitation of our work is that we do not assess the quality of the anchored edit distance for long reads compared to the other chaining methods, nor we integrate `llchain` in a seed-chain-extend workflow. Towards this direction, future work includes extending our method to the gap cost of A*PA [16], as well as a generalization that handles inversions. Furthermore, it should be possible to adapt the method to allow for arbitrary fragment scores, affine gap cost penalties, and inexact matches.

## References


1 Mohamed Ibrahim Abouelhoda and Enno Ohlebusch. CHAINER: Software for comparing genomes. In *Proceedings of the 12th International Conference on Intelligent Systems for Molecular Biology+ 3rd European Conference on Computational Biology*. Citeseer, 2004. Short paper. URL: `https://web.archive.org/web/20250708030638/https://www.iscb.org/cms_addon/conferences/ismbeccb2004/short%20papers/19.pdf`.

2 Mohamed Ibrahim Abouelhoda and Enno Ohlebusch. Chaining algorithms for multiple genome comparison. *J. Discrete Algorithms*, 3(2-4):321–341, 2005. `doi:10.1016/J.JDA.2004.08.011`.

3 Brenda S. Baker and Raffaele Giancarlo. Sparse dynamic programming for longest common subsequence from fragments. *J. Algorithms*, 42(2):231–254, 2002. `doi:10.1006/JAGM.2002.1214`.

4 Rudolf Bayer and Edward M. McCreight. Organization and maintenance of large ordered indexes. *Acta Informatica*, 1(3):173–189, 1972. `doi:10.1007/bf00288683`.

5 Gary Benson, Avivit Levy, S. Maimoni, D. Noifeld, and B. Riva Shalom. LCSk: A refined similarity measure. *Theor. Comput. Sci.*, 638:11–26, 2016. `doi:10.1016/J.TCS.2015.11.026`.

6 Gary Benson, Avivit Levy, and B. Riva Shalom. Longest common subsequence in $k$ length substrings. In *Similarity Search and Applications 2013*, pages 257–265. Springer Berlin Heidelberg, 2013. doi:10.1007/978-3-642-41062-8_26.

7 Elena Bushmanova, Dmitry Antipov, Alla L. Lapidus, Vladimir Suvorov, and Andrey D. Prjibelski. rnaQUAST: a quality assessment tool for *de novo* transcriptome assemblies. *Bioinform.*, 32(14):2210–2212, 2016. doi:10.1093/BIOINFORMATICS/BTW218.

8 Patrick S. G. Chain, Stefan Kurtz, Enno Ohlebusch, and Tom Slezak. An applications-focused review of comparative genomics tools: Capabilities, limitations and future challenges. *Briefings Bioinform.*, 4(2):105–123, 2003. doi:10.1093/BIB/4.2.105.

9 Kun-Mao Chao and Webb Miller. Linear-space algorithms that build local alignments from fragments. *Algorithmica*, 13(1/2):106–134, 1995. doi:10.1007/BF01188583.

10 Kun-Mao Chao, Jinghui Zhang, James Ostell, and Webb Miller. A local alignment tool for very long DNA sequences. *Comput. Appl. Biosci.*, 11(2):147–153, 1995. doi:10.1093/BIOINFORMATICS/11.2.147.

11 Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational geometry: algorithms and applications, 3rd Edition*. Springer, 2008. doi:10.1007/978-3-540-77974-2.

12 Arthur L. Delcher, Simon Kasif, Robert D. Fleischmann, Jeremy Peterson, Owen White, and Steven L. Salzberg. Alignment of whole genomes. *Nucleic Acids Research*, 27(11):2369–2376, January 1999. doi:10.1093/nar/27.11.2369.

13 Patrick Dinklage, Johannes Fischer, and Alexander Herlez. Engineering predecessor data structures for dynamic integer sets. In *19th International Symposium on Experimental Algorithms, SEA 2021, Nice, France, June 7-9, 2021*, volume 190, pages 7:1–7:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.SEA.2021.7.

14 David Eppstein, Zvi Galil, Raffaele Giancarlo, and Giuseppe F. Italiano. Sparse dynamic programming I: linear cost functions. *J. ACM*, 39(3):519–545, 1992. doi:10.1145/146637.146650.

15 David Eppstein, Zvi Galil, Raffaele Giancarlo, and Giuseppe F. Italiano. Sparse dynamic programming II: convex and concave cost functions. *J. ACM*, 39(3):546–567, 1992. doi:10.1145/146637.146656.

16 Ragnar Groot Koerkamp and Pesho Ivanov. Exact global alignment using A* with chaining seed heuristic and match pruning. *Bioinform.*, 40(1), 2024. doi:10.1093/BIOINFORMATICS/BTAE032.

17 Leo J. Guibas and Robert Sedgewick. A dichromatic framework for balanced trees. In *19th Annual Symposium on Foundations of Computer Science (sfcs 1978)*, pages 8–21, 1978. doi:10.1109/SFCS.1978.3.

18 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology*. Cambridge University Press, 1997. doi:10.1017/CBO9780511574931.

19 Yijie Han. Deterministic sorting in $O(n \log \log n)$ time and linear space. In John H. Reif, editor, *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 602–608. ACM, 2002. doi:10.1145/509907.509993.

20 Robert S. Harris. *Improved pairwise alignment of genomic DNA*. PhD thesis, Pennsylvania State University, December 2007. URL: https://www.bx.psu.edu/~rsharris/rsharris_phd_thesis_2007.pdf.

21 Chirag Jain, Daniel Gibney, and Sharma V. Thankachan. Algorithms for colinear chaining with overlaps and gap costs. *J. Comput. Biol.*, 29(11):1237–1251, 2022. doi:10.1089/CMB.2022.0266.

22 Donald B. Johnson. A priority queue in which initialization and queue operations take $O(\log \log D)$ time. *Mathematical Systems Theory*, 15(1):295–309, December 1981. doi:10.1007/bf01786986.

**23** Stefan Kurtz, Adam Phillippy, Arthur L. Delcher, Michael Smoot, Martin Shumway, Corina Antonescu, and Steven L. Salzberg. Versatile and open software for comparing large genomes. *Genome biology*, 5(2):R12, 2004. doi:10.1186/gb-2004-5-2-r12.

**24** Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinform.*, 34(18):3094–3100, 2018. doi:10.1093/BIOINFORMATICS/BTY191.

**25** Veli Mäkinen and Kristoffer Sahlin. Chaining with overlaps revisited. In Inge Li Gørtz and Oren Weimann, editors, *31st Annual Symposium on Combinatorial Pattern Matching, CPM 2020, Copenhagen, Denmark, June 17-19, 2020*, LIPIcs, pages 25:1–25:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.CPM.2020.25.

**26** Guillaume Marçais, Arthur L. Delcher, Adam M. Phillippy, Rachel Coston, Steven L. Salzberg, and Aleksey Zimin. Mummer4: A fast and versatile genome alignment system. *PLOS Computational Biology*, 14(1):e1005944, January 2018. doi:10.1371/journal.pcbi.1005944.

**27** Alla Mikheenko, Andrey D. Prjibelski, Vladislav Saveliev, Dmitry Antipov, and Alexey A. Gurevich. Versatile genome assembly evaluation with QUAST-LG. *Bioinform.*, 34(13):i142–i150, 2018. doi:10.1093/BIOINFORMATICS/BTY266.

**28** Gene Myers. An $O(ND)$ difference algorithm and its variations. *Algorithmica*, 1(1–4):251–266, November 1986. doi:10.1007/bf01840446.

**29** Gene Myers and Webb Miller. Chaining multiple-alignment fragments in sub-quadratic time. In Kenneth L. Clarkson, editor, *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, 22–24 January 1995. San Francisco, California, USA*, pages 38–47. ACM/SIAM, 1995. URL: http://dl.acm.org/citation.cfm?id=313651.313661.

**30** Gonzalo Navarro and Javiel Rojas-Ledesma. Predecessor search. *ACM Comput. Surv.*, 53(5):105:1–105:35, 2021. doi:10.1145/3409371.

**31** Sergey Nurk, Sergey Koren, Arang Rhie, Mikko Rautiainen, Andrey V. Bzikadze, Alla Mikheenko, Mitchell R. Vollger, Nicolas Altemose, Lev Uralsky, Ariel Gershman, Sergey Aganezov, Savannah J. Hoyt, Mark Diekhans, Glennis A. Logsdon, Michael Alonge, Stylianos E. Antonarakis, Matthew Borchers, Gerard G. Bouffard, Shelise Y. Brooks, Gina V. Caldas, Nae-Chyun Chen, Haoyu Cheng, Chen-Shan Chin, William Chow, Leonardo G. de Lima, Philip C. Dishuck, Richard Durbin, Tatiana Dvorkina, Ian T. Fiddes, Giulio Formenti, Robert S. Fulton, Arkarachai Fungtammasan, Erik Garrison, Patrick G. S. Grady, Tina A. Graves-Lindsay, Ira M. Hall, Nancy F. Hansen, Gabrielle A. Hartley, Marina Haukness, Kerstin Howe, Michael W. Hunkapiller, Chirag Jain, Miten Jain, Erich D. Jarvis, Peter Kerpedjiev, Melanie Kirsche, Mikhail Kolmogorov, Jonas Korlach, Milinn Kremitzki, Heng Li, Valerie V. Maduro, Tobias Marschall, Ann M. McCartney, Jennifer McDaniel, Danny E. Miller, James C. Mullikin, Gene Myers, Nathan D. Olson, Benedict Paten, Paul Peluso, Pavel A. Pevzner, David Porubsky, Tamara Potapova, Evgeny I. Rogaev, Jeffrey A. Rosenfeld, Steven L. Salzberg, Valerie A. Schneider, Fritz J. Sedlazeck, Kishwar Shafin, Colin J. Shew, Alaina Shumate, Ying Sims, Arian F. A. Smit, Daniela C. Soto, Ivan Sović, Jessica M. Storer, Aaron Streets, Beth A. Sullivan, Françoise Thibaud-Nissen, James Torrance, Justin Wagner, Brian P. Walenz, Aaron Wenger, Jonathan M. D. Wood, Chunlin Xiao, Stephanie M. Yan, Alice C. Young, Samantha Zarate, Urvashi Surti, Rajiv C. McCoy, Megan Y. Dennis, Ivan A. Alexandrov, Jennifer L. Gerton, Rachel J. O'Neill, Winston Timp, Justin M. Zook, Michael C. Schatz, Evan E. Eichler, Karen H. Miga, and Adam M. Phillippy. The complete sequence of a human genome. *Science*, 376(6588):44–53, April 2022. doi:10.1126/science.abj6987.

**32** Enno Ohlebusch. *Bioinformatics Algorithms: Sequence Analysis, Genome Rearrangements, and Phylogenetic Reconstruction*. Oldenbusch Verlag, 2013. URL: http://www.oldenbusch-verlag.de/.

**33** Enno Ohlebusch and Mohamed I Abouelhoda. Chaining algorithms and applications in comparative genomics. *Handbook of Computational Molecular Biology*, pages 15–1, 2006.

**34** Christian Otto, Steve Hoffmann, Jan Gorodkin, and Peter F. Stadler. Fast local fragment chaining using sum-of-pair gap costs. *Algorithms Mol. Biol.*, 6:4, 2011. doi:10.1186/1748-7188-6-4.

35 Filip Pavetic, Ivan Katanic, Gustav Matula, Goran Zuzic, and Mile Sikic. Fast and simple algorithms for computing both $LCS_k$ and $LCS_{k+}$. In Jan Holub and Jan Zdárek, editors, *Prague Stringology Conference 2018, Prague, Czech Republic, August 27-28, 2018*, pages 50–62. Czech Technical University in Prague, Faculty of Information Technology, Department of Theoretical Computer Science, 2018. URL: http://www.stringology.org/event/2018/p06.html.

36 Filip Pavetić, Goran Žužić, and Mile Šikić. *LCSk++*: Practical similarity metric for long strings. *arXiv*, 2014. doi:10.48550/arXiv.1407.2407.

37 Jingwen Ren and Mark J. P. Chaisson. lra: A long read aligner for sequences and contigs. *PLoS Comput. Biol.*, 17(6), 2021. doi:10.1371/JOURNAL.PCBI.1009078.

38 Arang Rhie, Sergey Nurk, Monika Cechova, Savannah J. Hoyt, Dylan J. Taylor, Nicolas Altemose, Paul W. Hook, Sergey Koren, Mikko Rautiainen, Ivan A. Alexandrov, Jamie Allen, Mobin Asri, Andrey V. Bzikadze, Nae-Chyun Chen, Chen-Shan Chin, Mark Diekhans, Paul Flicek, Giulio Formenti, Arkarachai Fungtammasan, Carlos Garcia Giron, Erik Garrison, Ariel Gershman, Jennifer L. Gerton, Patrick G. S. Grady, Andrea Guarracino, Leanne Haggerty, Reza Halabian, Nancy F. Hansen, Robert Harris, Gabrielle A. Hartley, William T. Harvey, Marina Haukness, Jakob Heinz, Thibaut Hourlier, Robert M. Hubley, Sarah E. Hunt, Stephen Hwang, Miten Jain, Rupesh K. Kesharwani, Alexandra P. Lewis, Heng Li, Glennis A. Logsdon, Julian K. Lucas, Wojciech Makalowski, Christopher Markovic, Fergal J. Martin, Ann M. Mc Cartney, Rajiv C. McCoy, Jennifer McDaniel, Brandy M. McNulty, Paul Medvedev, Alla Mikheenko, Katherine M. Munson, Terence D. Murphy, Hugh E. Olsen, Nathan D. Olson, Luis F. Paulin, David Porubsky, Tamara Potapova, Fedor Ryabov, Steven L. Salzberg, Michael E. G. Sauria, Fritz J. Sedlazeck, Kishwar Shafin, Valery A. Shepelev, Alaina Shumate, Jessica M. Storer, Likhitha Surapaneni, Angela M. Taravella Oill, Françoise Thibaud-Nissen, Winston Timp, Marta Tomaszkiewicz, Mitchell R. Vollger, Brian P. Walenz, Allison C. Watwood, Matthias H. Weissensteiner, Aaron M. Wenger, Melissa A. Wilson, Samantha Zarate, Yiming Zhu, Justin M. Zook, Evan E. Eichler, Rachel J. O'Neill, Michael C. Schatz, Karen H. Miga, Kateryna D. Makova, and Adam M. Phillippy. The complete sequence of a human Y chromosome. *Nature*, 621(7978):344–354, August 2023. doi:10.1038/s41586-023-06457-y.

39 Nicola Rizzo, Manuel Cáceres, and Veli Mäkinen. Practical colinear chaining on sequences revisited. In Jing Tang, Xin Lai, Zhipeng Cai, Wei Peng, and Yanjie Wei, editors, *Bioinformatics Research and Applications - 21st International Symposium, ISBRA 2025, Helsinki, Finland, August 3-5, 2025, Proceedings, Part II*, volume 15757 of *Lecture Notes in Computer Science*, pages 203–216. Springer, 2025. doi:10.1007/978-981-95-0695-8_17.

40 Nicola Rizzo, Manuel Cáceres, and Veli Mäkinen. Practical colinear chaining on sequences revisited. *arXiv*, abs/2506.11750, 2025. doi:10.48550/arXiv.2506.11750.

41 Nicola Rizzo and Ragnar Groot Koerkamp. nrizzo/llchain. Software, EU Horizon Europe 101060011, swhId: swh:1:dir:78de8057c7f61afc5518ef3b8a46a5ac91e21d52 (visited on 2026-08-15). URL: https://github.com/nrizzo/llchain, doi:10.4230/artifacts.27683.

42 Kristoffer Sahlin, Thomas Baudeau, Bastien Cazaux, and Camille Marchet. A survey of mapping algorithms in the long-reads era. *Genome Biology*, 24(1):133, 2023.

43 Kristoffer Sahlin and Veli Mäkinen. Accurate spliced alignment of long RNA sequencing reads. *Bioinform.*, 37(24):4643–4651, 2021. doi:10.1093/BIOINFORMATICS/BTAB540.

44 P. van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Information Processing Letters*, 6(3):80–82, June 1977. doi:10.1016/0020-0190(77)90031-x.

45 W John Wilbur and David J Lipman. Rapid similarity searches of nucleic acid and protein data banks. *Proceedings of the National Academy of Sciences*, 80(3):726–730, 1983.

46 W John Wilbur and David J Lipman. The context dependent comparison of biological sequences. *SIAM Journal on Applied Mathematics*, 44(3):557–567, 1984.

47 Dan E. Willard. Log-logarithmic worst-case range queries are possible in space $\Theta(N)$. *Information Processing Letters*, 17(2):81–84, August 1983. doi:10.1016/0020-0190(83)90075-3.

48 Zheng Zhang, Balaji Raghavachari, Ross C. Hardison, and Webb Miller. Chaining multiple-alignment blocks. *J. Comput. Biol.*, 1(3):217–226, 1994. doi:10.1089/CMB.1994.1.217.

# Quik 2.0: Efficient Large-Scale DNA Barcode Calling

**Steffen Schüler**
Martin Luther University Halle-Wittenberg, Germany

**Antonia Schmidt**
Martin Luther University Halle-Wittenberg, Germany

**Matthias Müller-Hannemann**
Martin Luther University Halle-Wittenberg, Germany

## Abstract

DNA barcodes are used as unique identifiers in high-throughput sequencing technologies with applications in areas such as single cell analysis, spatial transcriptomics and DNA data storage. Given a set of barcodes and a set of reads, each containing a barcode, the task of barcode calling is to assign each read to its respective barcode. This is challenging in applications involving large barcode sets and high rates of base insertion, deletion and substitution errors. Naive solutions require the calculation of pairwise distances between each barcode and read. As this is infeasible for modern applications with millions of barcodes and billions of reads, much work has been done during the previous years in accelerating this task. In 2026, Uphoff et al. introduced the barcode calling tool *Quik* based on $k$-mer filtering and pseudo-distances. They demonstrated that it is faster than state-of-the-art tools by several orders of magnitude. Here, we present *Quik* 2.0, which is faster than the original release by a factor of up to 56 and scales well to multiple GPUs. We discuss several algorithmic design choices that led to this speedup. In large-scale experiments with $10^6$ barcodes, we can now process approximately 300 million reads per hour on a GPU server equipped with four GPUs. In addition, we show that unfiltered barcode calling approaches can only slightly improve the accuracy at the cost of a vastly increased running time. Finally, we introduce more fine-grained assignment rejection criteria to achieve a better trade-off between precision and acceptance rate. To help users select suitable rejection parameters for real-world experiments, we propose an automatic calibration procedure that optimizes parameters for specific barcode and read sets.

**2012 ACM Subject Classification** Applied computing → Bioinformatics; Computing methodologies → Parallel algorithms; Applied computing → Computational genomics

**Keywords and phrases** DNA barcode calling, $k$-mer filtering, GPU computing, algorithm engineering, spatial transcriptomics

**Digital Object Identifier** 10.4230/LIPIcs.WABI.2026.14

**Supplementary Material** *Software (Source Code)*: `https://github.com/uni-halle/quik` [20] archived at `swh:1:dir:948d8ae0e0b66260e91d201ea0b5f6f732aa4eff`

## 1 Introduction

DNA barcodes are short unique sequences of nucleotides added to DNA fragments to identify samples in pooled sequencing experiments. They have become an essential tool in modern genomics, supporting applications such as single-cell analysis [10], spatially resolved transcriptomics [13, 23], and emerging technologies like DNA-based information storage [2]. Barcodes can encode samples in multiplex experiments, cell lines for single-cell transcriptomics or locations in spatial transcriptomics experiments.

Advances in sequencing technologies now allow experiments to generate vast numbers of reads, each associated with a barcode. In practice, these barcodes are rarely observed without errors: synthesis imperfections and sequencing artifacts introduce substitutions, insertions, and deletions. Recovering the original barcode from such corrupted observations is known


© Steffen Schüler, Antonia Schmidt, and Matthias Müller-Hannemann;
licensed under Creative Commons License CC-BY 4.0

as *barcode calling* and constitutes a key step in downstream analysis. From an algorithmic perspective, barcode calling can be formulated as a nearest-neighbor search problem with respect to a suitable distance measure on DNA sequences. The basic algorithmic task is as follows: Given a set of barcodes and a set of reads, determine a barcode of minimum distance for each read.

**Related Work.** If the expected error rate is very low, one can simply look for exact barcode matches, which is very fast and often used in practice. Error-correcting codes such as those from Hawkins et al. [9] also allow fast barcode calling, but impose severe limitations on the number of barcodes and the error rate. Modern techniques such as photolithographic microarray synthesis enable the synthesis of millions of barcodes needed for spatial transcriptomics studies at reasonable cost [4, 15]. However, their error rates may exceed 20% per base, as reported by Lietard et al. [12]. This makes naive approaches inapplicable. Recent work has shown that randomly generated barcode sets of adequate length can remain distinguishable even under substantial noise, suggesting new design principles for large-scale experiments with practically unlimited barcode sets [18].

As explained above, the central idea of many barcode calling approaches is to determine for each read a *most similar* barcode with respect to a given distance measure such as the Hamming distance [6, 14], the edit or Levenshtein distance [11, 1, 8], or the Sequence-Levenshtein (SL) distance [5]. The latter is similar to the Levenshtein distance, but allows either sequence to be truncated at the end without penalty. While the Hamming distance does not account for insertions or deletions ("indels") in the read, the Levenshtein distance and the SL distance do so. In general, the basic strategy is to use an all-to-all distance computation to identify the best match. Tools such as Dorado [16], Flexiplex [7] and Barbell [3] perform all-to-all distance computations for demultiplexing barcodes in Oxford Nanopore sequencing applications, in which barcode sets are usually small.

However, directly comparing every read against all candidate barcodes is computationally prohibitive in applications with millions of barcodes and billions of reads. This has led to the development of more scalable approaches, including data structures for efficient distance computation, clustering-based heuristics, and multi-stage filtering techniques that prune the search space before applying more expensive comparisons. Zorita et al. [24] propose the calculation of all pairwise edit distances using prefix trees and several optimizations of the classical Needleman-Wunsch algorithm as well as clustering ("starcode"). Tambe and Pachter [21] develop the barcode-calling approach "sircel" and demonstrate its efficacy for small barcode sets and short barcodes.

To cope with large barcode sets, Press introduced RandomBarcodes [18], which implements a filtering approach called triage that identifies promising barcodes by considering the occurrence and corresponding positions of trimers. Uphoff et al. developed Quik [22], which pre-filters the barcodes using a $k$-mer filter. Both tools achieve a significant speedup compared to exhaustive all-to-all comparisons and run on GPUs. Columba, a general-purpose read aligner developed by Renders et al. [19], was not designed for barcode calling, yet has proven effective for this purpose. It implements an exact search for hits with up to 13 errors sped up by clever search schemes and performance enhancement techniques.

Despite these improvements, achieving both high accuracy and computational efficiency remains a significant challenge, particularly in scenarios involving millions of barcodes and high error rates. A recent benchmark study by Poma-Soto et al. [17] evaluated barcode calling tools for such scenarios, including RandomBarcodes, Quik and Columba. It concluded that Quik outperforms the other tools in terms of speed, accuracy and scalability on real sequencing data from spatial transcriptomics.

**Contribution.** In this paper, we introduce Quik 2.0, which significantly improves upon the original version. Our achievements are as follows.

- First, we carefully reengineer the $k$-mer based filtering approach of Quik to better exploit the capabilities of GPUs and enable multi-GPU support. In large-scale experiments with $10^6$ barcodes, we achieve a speed-up factor of 16 on a single GPU and a speed-up factor of 56 on a GPU server with four GPUs. On such a GPU server, we reach a throughput of more than 80,000 reads per second.
- Second, we provide an implementation without filtering, i.e., a version which exhaustively compares each read with each barcode. Unsurprisingly, this unfiltered barcode calling method is orders of magnitudes slower than the filtering approaches. In fact, we observed that it is more than 500 times slower in our experiments. However, this slow method is still useful to validate the filtering approaches. It turns out that the unfiltered method has only a slightly better trade-off between precision and acceptance rate.
- Third, we reconsider the final phase of barcode calling. In the original version of Quik, the user has to choose an application-dependent rejection parameter $D$: if the distance of a read to the barcode candidate exceeds the threshold $D$, this read is rejected. Such a rejection rule is necessary since in practice some reads may be corrupt due to high error rates or experimental artifacts making it impossible to detect a barcode. In this work, we revisit this simple rejection criterion and propose several more sophisticated alternatives. In particular, we consider not only the distance to the best candidate, but also the distance to the second best. Based on these two distance values, we derive more complex rejection criteria allowing more fine-grained trade-off between precision and acceptance rate.

  In practice, the choice of appropriate rejection criteria highly depends on the barcode and read set and is thus nontrivial. We propose a method to flexibly and automatically calibrate suitable rejection criteria for each data set. In large-scale synthetic experiments, we find that more sophisticated rejection criteria and their optimization can improve the trade-off between precision and acceptance rate.

**Structure.** The remainder of this paper is structured as followed: In the following Section 2, we revisit essential ideas of Quik [22], at a level sufficient to understand the algorithmic principles. In Section 3, we discuss several algorithmic improvements and present associated speedups. Afterwards, in Section 4, we discuss how much the unfiltered calling approach can improve the accuracy of the barcode calling at the cost of a vastly increased running time. In Section 5, we present our method to automatically determine criteria to reject falsely assigned barcodes. Finally, in Section 6, we summarize our findings.

## 2 Quik Barcode Calling

In this section, we briefly summarize essential ideas of Quik as introduced by Uphoff et al. [22]. Quik requires two files including $n$ barcodes and $m$ reads, respectively. While the barcodes are assumed to have a fixed length $\ell$, the read lengths may vary. Quik assumes, however, that the reads have already approximately been trimmed to the relevant area in which the barcode is located.

**Preprocessing.** Quik first applies a preprocessing of the barcodes. Depending on a parameter $k$, the length of $k$-mers, it initially constructs subsets $L(x, j)$ of the barcode indices $1, \ldots, n$. For each possible $k$-mer $x \in \{A, C, G, T\}^k$ and $k$-mer start position $j \in$

$\{0, \ldots, \ell - k\}$ within a barcode, the associated subset $L(x, j)$ contains exactly the indices of the barcodes in which $x$ occurs as a substring starting at position $j$. This preprocessing can be done efficiently in a single sweep over the barcodes.

**Main loop.** In its original design, Quik processes the reads independently of each other in parallel, associating *exactly one* CUDA thread to each read. Each thread *sequentially* runs the following steps:

1. First, it computes a so-called *pseudo-distance* between its associated read $r$ and all barcodes (see Algorithm 1). This pseudo-distance is always non-positive, with zero as the largest possible value. The smaller the value of the pseudo-distance, the more similar is the respective barcode to the read. Hence, we are only interested in barcodes with *negative* pseudo-distance. Thus, Algorithm 1 compiles and returns a set $C$ of all barcode indices with negative pseudo-distance, together with the array $P$ of all pseudo-distances.
2. Next, each thread identifies from the candidate set $C$ the $c = 100$ barcodes with minimum pseudo-distance to $r$. This has been implemented by a single sweep over $C$ and insertion-sorting the barcodes into a reduced candidate list $C'$ of at most $c$ elements, ordered by their pseudo-distance. As $c$ is very small, the new list $C'$ can be stored in registers that are private to each CUDA thread. In doing so, the construction of $C'$ is sufficiently fast and requires only a single sweep over the candidate set $C$.
3. In the third step, each thread calculates the SL distance between its associated read $r$ and the $c$ barcodes in its reduced candidate list $C'$. There are several approaches to calculate the SL distance. As the barcode length $\ell = 34$ is very small in our application, we found that the fastest way is the classical dynamic programming approach, which runs $\Theta(\ell^2)$ steps and uses $\Theta(\ell)$ extra registers. From the candidate barcodes of $C'$, we select one with *minimum* SL distance and assign it to $r$. In case of a tie, we assign $r$ to a barcode with smaller pseudo-distance.
4. Finally, each thread evaluates some rejection criterion to decide whether to accept the assignment of the associated read $r$ to the previously selected barcode with minimum SL distance, or to reject it. In the original implementation of Quik, a read $r$ is *rejected* if its SL distance is larger than some predefined rejection threshold $D$ given by the user.

**Problems and imperfections of Quik 1.0.** Although Uphoff et al. demonstrated the superiority of their approach to that of Press, the original design of Quik as described above was not explicitly designed to run on GPUs. In particular, the following aspects within the first step of the main loop are suboptimal for a GPU implementation:

1. As the arrays $P$ of pseudo-distances have $n$ elements each (and $n$ is about one million in our application), these arrays do *not* fit into the private or shared GPU memory spaces and must thus be stored in the *global* memory of the GPU. As each thread writes to its own array $P$ and therefore to a *different* area of the global memory, the writing accesses to $P$ in Line 11 of Algorithm 1 are uncoalesced and thus slow, as individual memory requests from different threads cannot be combined.
2. A similar problem occurs with the arrays $C$ of candidate barcodes. These must also be stored in the global memory, as the number of barcodes with negative pseudo-distance cannot be bounded by a small constant in advance. As a consequence, accesses to these arrays are uncoalesced, too.

**Algorithm 1** The original algorithm for the computation of the $k$-mer pseudo-distance between a fixed read $r$ and all barcodes as presented by Uphoff et al. [22].

**Input:** Read $r$, barcode count $n$, barcode lists $L(x, j)$
**Output:** Candidate set $C \subseteq \{1, \ldots, n\}$ and array $P \in \mathbb{Z}^n$ of pseudo-distances
**Parameters:** Length $\ell$ of barcodes and reads, length $k$ of $k$-mers, $d_{\max} = 5$

```
1  P ← 0^n        // pseudo-distance array of length n, initialized with zero
2  C ← ∅          // indices of barcodes with nonzero pseudo-distance
3  for each possible k-mer starting position i ∈ {0, ..., |r| − k} in the read r do
4      x ← r[i..i + k − 1]                          // k-mer starting at position i
5      j_start ← max(0, i − d_max)
6      j_end ← min(ℓ − k, i + d_max)
7      for each possible k-mer starting position j ∈ {j_start, ..., j_end} do
8          for each barcode index b ∈ L(x, j) do
9              if P[b] = 0 then
10                 C ← C ∪ {b}
11             P[b] ← P[b] + |i − j| − ℓ
12 return C and P
```

3. As each CUDA thread sequentially processes the lists $L(x, j)$ in Line 8 of Alg. 1 (also stored in the global GPU memory due to their size), the threads of the same CUDA block again access *different* memory locations in an uncoalesced way.
4. Furthermore, as the individual lists $L(x, j)$ have different lengths, the CUDA threads of the same CUDA block suffer from thread divergence and are limited by the length of the longest list within the CUDA block.

In summary, a main problem of the original Quik implementation were a lot of uncoalesced (and thus slow) accesses to the global memory.

## 3 Algorithmic Improvements

To overcome these limitations, we made several adjustments to the original algorithm, from which we are going to present some instructive ones. For each improvement step, we performed a benchmark experiment to evaluate its effect on Quik's total running time. The speedup techniques discussed here are specific for Quik but follow general principles that can be summarized as follows:

- Exploit the shared GPU memory whenever you can.
- When accessing the global GPU memory, use a coalesced memory access pattern. In doing so, memory requests from threads are combined into a small number of large, efficient memory transactions, thereby making better use of the available memory bandwith.
- Make sure that all threads have enough work.

Although these principles are well-known in the context of GPU computing, it is not yet clear how to apply them to the algorithms used in Quik.

**Benchmark experiment.** To assess the effect of the optimization steps, we re-ran the benchmark experiment as described by Uphoff et al [22]. In this experiment, a set of $n = 10^6$ random barcodes of length $\ell = 34$ is used, as Press showed that barcode sets of this size

and length remain distinguishable even under high error rates [18]. From these barcodes, $m = 10^6$ synthetic reads, each also of length $\ell = 34$, are generated with an error probability of 20% per base, distributed evenly between substitutions, insertions, and deletions.

In our benchmark experiment, we used Quik's 4-mer filter approach to assign the reads to the barcodes. Quik offers other filtering approaches but Uphoff et al. demonstrated that $k = 4$ gives the highest accuracy. For this reason, we focused on optimizing the 4-mer filter. We ran the program on a Linux system with four AMD EPYC 7662 64-Core processors (thus giving a total of 256 logical CPU cores), 1 TB of main memory, and four NVIDIA A100-SXM4 GPUs, each having 6912 CUDA cores and 40 GB of GPU memory. As the original version of Quik does not support the use of multiple GPUs, only one of them is used until the very last step. For each optimization step, we determined the average wall clock time of three independent runs as our performance measure.

**Step 1: Omit the candidate set $C$.** As discussed in the previous section, the construction of the barcode set $C$ with negative pseudo-distance induces an uncoalesced memory access pattern. In fact, our benchmark experiment showed that it is faster to omit the construction of $C$ and, in the second step of the main loop, to scan *all* barcodes while identifying the ones with smallest pseudo-distance. This gives a speedup factor of about 2.17 to the original version of Quik.

**Step 2: Introducing parallelism at the read level.** Instead of assigning a single thread to each read, we associate with each read a whole *block* of threads. The exact number $t$ of threads per block is a parameter that can be optimized for the fine tuning of the algorithm. In the following steps, we experimented with different values of $t$ ranging from 32 to 256 and individually selected a value of $t$ that minimizes the running time. Introducing multiple threads per read gives us the opportunity to parallelize various parts of the main loop:

- In the first phase of the main loop, we parallelize Line 8 of Algorithm 1 by evenly distributing the elements from $L(x, j)$ between the $t$ threads of the same block. These threads can now read *adjacent* elements from $L(x, j)$, leading to a coalesced (and thus faster) memory access pattern.
- There are many ways to parallelize the second phase of the main loop, in which we need to construct a reduced candidate set $C'$ of $c = 100$ barcodes with smallest pseudo-distances. For the moment, we stay with the sequential insertion-sort-like construction scheme as described above, only using one thread of each block. We will discuss techniques for parallelization in a few moments.
- In the third phase of the main loop, we parallelized the computation of the SL distances by distributing the candidate barcodes from $C'$ evenly between the threads such that each thread computes about $c/t$ SL distances. As all threads of the same block perform *exactly* the same computations on different data, this is ideal for SIMD processing units like GPUs.

Implementing this step actually slows down the running time by a factor of 0.73 (relatively to the previous step) but is a necessary prerequisite to exploit the shared memory in the following steps.

**Step 3: Exploiting the shared memory.** The original version of Quik does not make use of the shared GPU memory, which is known to be several times faster than the global memory, but is very limited in size (typically only about 100 KB per shared multiprocessor). As discussed above, the hindering problem in storing the pseudo-distance array $P$ in the shared memory is the large number $n$ of barcodes in our application.

To overcome this problem, we partition the barcodes into chunks of $n' < n$ barcodes, and process these chunks sequentially. For each chunk, the threads associated to a read $r$ compute the pseudo-distances between $r$ and just the barcodes of the current chunk. Choosing $n'$ as a small constant (we used $n' = 2048$), the pseudo-distance array $P$ easily fits into the shared memory. There are several consequences to this modification:

- Accessing the pseudo-distance array $P$ is now *much* faster.
- The precomputed lists $L(x, j)$ used to compute the pseudo-distances must now be restricted to the individual barcode chunks. Thus, in the preprocessing step, we now compute a whole collection of lists $L(x, j)$, one for each chunk.
- In contrast to the general description of the main loop above, we now cannot strictly separate the first two phases. Instead, these phases are interleaved: Each block of threads maintains its candidate list $C'$ of at most $c$ elements, which is updated while processing the barcode chunks.

Implementing this idea gives a speedup of 2.32 relatively to the previous step, now leading to an absolute speedup factor of $2.17 \times 0.73 \times 2.32 = 3.67$ in comparison to the original version.

**Step 4: Parallel update of the candidate lists.** In the second step of the main loop, we must identify the $c = 100$ barcodes with smallest pseudo-distance. As discussed above, an insertion sort like approach has been used to create a reduced candidate list $C'$ of barcode indices, ordered by their pseudo-distance. So far, the creation of this list has been done sequentially by one thread of each CUDA block, while all other threads are stalled. However, we can easily use a parallel approach to insert a barcode $b$ with pseudo-distance $q$ into the candidate list $C'$:

- First, all threads of the same block use (sequential) binary search to identify a position $p \in \{1, \ldots, c\}$ within $C'$ to which the new element needs to be inserted to keep the barcode indices ordered by their pseudo-distance. This needs additional $\Theta(\log c)$ steps, which is a very small constant as $c \leq 100$.
- Second, all threads of the same block in parallel move the elements of the sub-list $C'[p, c]$ one position to the right, thereby making room for the new element $(b, q)$ at position $p$. As threads of the same warp perform in lock-step, this needs only very few accesses to registers.
- Finally, one thread of each block inserts the new element $(b, q)$ at position $p$ of $C'$.

Implementing this idea gives a small speedup of 1.09 in comparison to the previous step and thus an absolute speedup of $2.17 \times 0.73 \times 2.32 \times 1.09 = 4.01$.

**Step 5: Simultaneous access to multiple lists.** As a drawback of breaking the barcodes into small chunks, the lists $L(x, j)$ of barcode indices associated to each chunk are often very short (only containing a handful of barcode indices). This severely limits the performance when traversing these lists in Line 8 of Algorithm 1.

To overcome this problem, the threads of each block now access *multiple* lists $L(x, j)$ at once. For this purpose, we concatenated the lists $L(x, j)$ in ascending order of $j$ into one large list $L(x)$. When creating this list, we augment each element of $b \in L(x, j)$ by the associated starting position $j$. Thus, $L(x)$ is a long list of pairs $(b, j)$ of barcodes $b$ (more precisely, their indices) and starting position $j$ such that for each element $(b, j) \in L(x)$ the $k$-mer $x$ is a substring of the barcode $b$ starting at position $j$. For each value of $j$, we additionally store the start and end position of the original list $L(x, j)$ within $L(x)$. Thus, we can efficiently identify the start and end positions of each subset $L(x, j)$ within $L(x)$.

**Algorithm 2** The algorithm for the computation of the $k$-mer pseudo-distance between a fixed read $r$ and all barcodes of a chunk as described in Step 5.

**Input:** Read $r$, barcode count $n'$, barcode lists $L(x, j)$
**Output:** Array $P \in \mathbb{Z}^{n'}$ of pseudo-distances

$P \leftarrow 0^{n'}$
**for each** possible $k$-mer starting position $i \in \{0, \ldots, |r| - k\}$ in the read $r$ **do**
  $x \leftarrow r[i..i + k - 1]$
  $j_{\text{start}} \leftarrow \max(0, i - d_{max})$
  $j_{\text{end}} \leftarrow \min(\ell - k, i + d_{max})$
  $L' \leftarrow$ sub list from $L(x)$ associated to $L(x, j_{\text{start}})$ to $L(x, j_{\text{end}})$
  **for each** $(b, j) \in L'$ **in parallel do**
    $P[b] \leftarrow P[b] + |i - j| - \ell$ `// Atomic Add!`
**return** $P$

Algorithm 2 formalizes and implements this idea, which leads to a speedup of 4.02 relative to the previous step, and to a total speedup of $2.17 \times 0.73 \times 2.32 \times 1.09 \times 4.02 = 16.10$.

**Step 6: Use multiple GPUs.** Finally, we make use of multiple GPUs by evenly distributing the reads among the GPUs. Although this is no trivial task from the perspective of software design, it is still algorithmically straightforward and gives a speedup that is almost linear to the number of GPUs. In our benchmark experiment with 4 GPUs, we found a speedup of about 3.5, leading to a total speedup of $2.17 \times 0.73 \times 2.32 \times 1.09 \times 4.02 \times 3.50 = 56.35$ to the original version of Quik.

## Experimental Results

We repeated our benchmark experiment for other values of $k \in \{5, 6, 7\}$, and for the two-step approaches described by Uphoff et al. [22]. These approaches first use a 7-mer filter to assign reads to barcodes, and afterwards run a 4-mer (resp. 5-mer) filter on the subset of the still unassigned reads. Uphoff et al. showed that these approaches give similar accuracy as the pure 4-mer (resp. 5-mer) filters, while lowering the running time.

Table 1 shows the running time and resulting speedup factors for Quik's filtering variants after our optimization. We want to highlight some important observations:

- The 4-mer and 5-mer filter now have the lowest running times from all filtering variants. This stands in contrast to the observations of Uphoff et al., where it was shown that the running time of the original version of Quik declines with growing values of $k$.
- While profiting from the algorithmic optimization, all of the pure $k$-mer filter approaches are now faster than the two-step approaches. This is due to the additional overhead of identifying the unassigned reads and re-compiling them compactly. Again, this is in contrast to the observations from Uphoff et al.
- Unfortunately, the speedup induced by our optimization steps declines rapidly with growing values of $k$. However, that doesn't have to concern us, since the 4-mer filter is the one with the largest accuracy (which has been demonstrated by Uphoff et al., and which will also be shown later in Table 2). Thus, the 4-mer filter is our primary concern.

We conclude that the 4-mer filter is now the best filtering variant as it gives the highest accuracy while also being among the fastest.

**Table 1** Total running time (in seconds) for our benchmark experiment, processing $10^6$ reads. Speedup (relative to the original version of Quik as described by Uphoff et al.), and throughput (in reads per second) for the different filtering modes of Quik.

| Filter | Runtime | Speedup | Throughput |
|---|---|---|---|
| 4-mer | 11.7 | 56.35 | 83,097 |
| 5-mer | 10.8 | 20.95 | 92,712 |
| 6-mer | 14.8 | 5.48 | 65,167 |
| 7-mer | 18.8 | 2.29 | 53,204 |
| 7+4-mer | 24.4 | 2.28 | 40,386 |
| 7+5-mer | 25.5 | 3.72 | 41,326 |

## 4 Unfiltered Barcode Calling

Quik's key idea as presented by Uphoff et al. was to avoid expensive all-to-all SL computations between reads and barcodes by identifying a small candidate set $C'$ of at most $c = 100$ barcodes, and to restrict the SL computations to this set. In doing so, the total number of SL computations is reduced from $mn$ to $cm$ and thus to $\Theta(m)$. However, if the true barcode is not among the $c$ candidates, there is no chance to correctly assign the associated read.

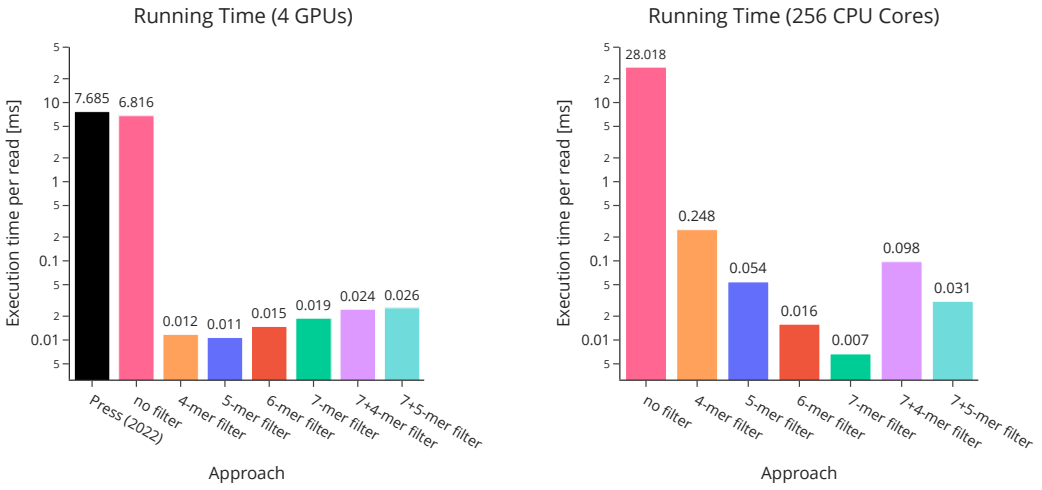
This motivates us to find out how much more precise our read assignment might be if we did not use a filter at all. In this section, we present the *unfiltered* calling approach, which does not make use of the pseudo-distance filter but uses an efficient way of all-to-all SL computations. We show how much this improves Quik's accuracy and study how much running time it costs.

### Algorithmic Approach

We start by giving a few ideas on how to implement the all-to-all SL computations *efficiently*. Like the $k$-mer filter approaches of Quik, the unfiltered barcode calling approach is designed to run efficiently on GPUs.

- We assign to each read $r$ a block of $t = 64$ CUDA threads, between which we distribute the barcodes evenly. Each thread then *sequentially* computes the SL distances between $r$ and its subset of about $n/t$ barcodes. Similar as discussed in the previous section, all threads run exactly the same sequence of operations on different data, which is perfect for GPUs.
- During the computations of the SL distances, each thread stores within its private memory the index of a barcode with minimum SL distance from its subset. After all threads have computed their minimum SL distance, we use an in-block reduction to determine the minimum SL distance within $\Theta(\log t)$ additional steps.
- We optimize the access to the global memory by using a carefully designed data structure for the barcodes that allows coalesced memory accesses when reading the barcodes.

We gain an algorithm that runs a total number $mn$ of SL computations, each requiring $\Theta(\ell^2)$ sequential operations. In contrast, the filtered approaches require only $cm$ SL computations in the third phase of the main loop. Thus, the total number of SL computations when using no filter is increased by a factor of $n/c$. In our benchmark experiment with $n = 1,000,000$ and $c = 100$, this gives us a rough estimation of a $10,000$-fold increase of the running time when omitting the filter.




**(a)** Running time using 4 NVIDIA A100 GPUs. **(b)** Running time using 4 AMD EPYC 7662 CPUs.

**Figure 1** Normalized running time (ms per read) of Quik's filtering approaches in our benchmark experiment. The ordinates use a logarithmic scale. In Figure 1b, the bar for Press is missing as his program does not work without a GPU.

## Running Time on GPUs

To assess the running time of our unfiltered barcode calling approach in practice, we repeated the benchmark experiment from the previous section. We compared the running time of the unfiltered approach with Quik's $k$-mer based filtering approaches, and with the filtering approach of Press (2022) [18]. Figure 1a shows the results.

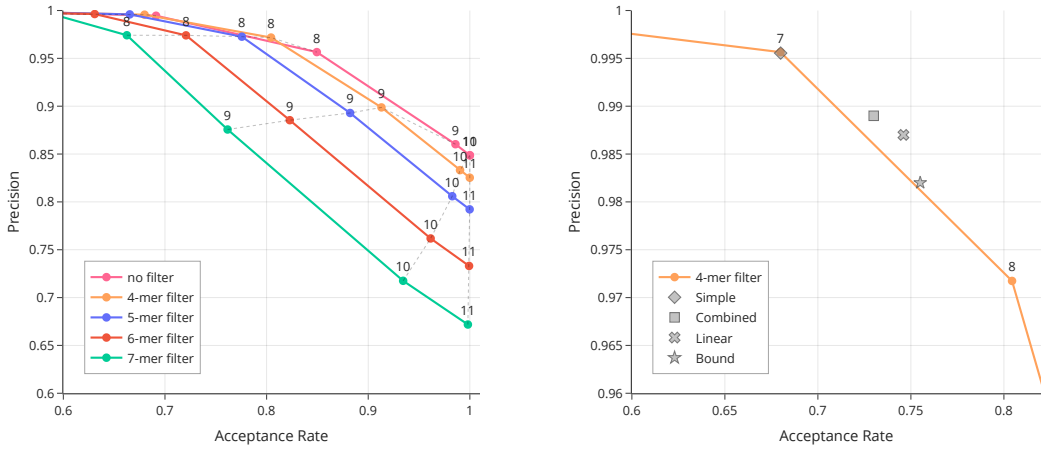
We want to highlight some essential observations.

- The unfiltered calling approach is in practice slower than the 4-mer filter by a factor of 568, which fortunately is far less than the slowdown factor of 10,000 discussed above. This is due to the fact that a significant proportion of running time in Quik's $k$-mer filtering approaches is used for the first and second phase of the main loop, which we do not have in the unfiltered approach.
- To our greatest surprise, our unfiltered approach is even slightly *faster* than the Random-Barcodes filtering approach suggested by Press (2022). From our perspective, this is due to Press' non-optimal parallelization strategy, which processes the reads *sequentially* and only parallelizes the search for their most similar barcodes. In contrast, Quik processes the *reads* in parallel.

Our experiment demonstrates that a careful GPU implementation of all-to-all SL computations can be a feasible option in practical applications, even with millions of barcodes and reads.

## Running Time on Multi-Core CPUs

For systems without a GPU, Quik offers a fallback mode that uses OpenMP to enable classical CPU parallelism. For a better interpretation of the GPU running times discussed so far, we repeated the benchmark experiment on our multi-core CPU using 256 OpenMP threads. The results are shown in Figure 1b.




**(a)** Comparison of the filter strategies for different rejection thresholds $D$.

**(b)** Zoomed view: 4-mer-filter with optimized rejection functions from Section 5.

**Figure 2** Precision vs acceptance rate for different values of the threshold distance $D$ on our benchmark experiment. For each filtering approach and each value of $D \in [7, 11]$, we gain one point. The points associated to the same value of $D$ are connected via a dashed line.

- Concerning the $k$-mer filtering approaches, we confirm the findings from Uphoff et al.: The running time of the pure $k$-mer filters steadily decreases with growing $k$, and the two-step filter approaches are still faster than the pure 4-mer and 5-mer filters.
- Using GPUs, the 4-mer filter is about 20 times faster than its CPU equivalent. This speedup declines with growing values of $k$. For $k = 7$, it would in fact be better to use the CPU implementation. This is consistent with our previous findings, as we saw that the speedup effect of the algorithm optimization declines with $k$.
- The unfiltered calling approach is about 4.1 times slower on the CPU system.

Our observations indicate that the GPU implementation of Quik is in most situations faster and thus more relevant in practical applications. However, in applications where the limited accuracy of the 7-mer filter is sufficient, it might in fact be faster to run Quik on classical multi-core CPUs.

## Accuracy

To analyze how much the accuracy of the barcode calling profits from using all-to-all distance computations, we repeated the synthetic experiment from Uphoff et al. In this experiment, the authors used different values of $D$ to reject reads if their SL distance is larger than $D$. Varying the rejection threshold D allows the user to control the number of accepted reads and, consequently, the accuracy of the resulting barcode assignment. Generally, the larger the chosen rejection threshold, the more reads will be accepted and the larger will be the number of accepted reads assigned to a wrong barcode.

To quantify the accuracy of Quik's calling approaches, we calculated two metrics: the *acceptance rate*, i.e. the number of accepted reads divided by the total number of reads, and the *precision*, i.e. the number of correctly accepted reads divided by the total number of accepted reads. Note that the precision can only be calculated in synthetic experiments with a known ground truth, whereas the acceptance rate can also be calculated in real-life

**Table 2** Acceptance rate and precision (in percent) for each rejection threshold $D \in [7, 11]$.

| | Acceptance rate | | | | | Precision | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Approach | 7 | 8 | 9 | 10 | 11 | 7 | 8 | 9 | 10 | 11 |
| no filter | 69.1 | 84.9 | 98.6 | 99.9 | 100 | 99.5 | 95.7 | 86.0 | 84.9 | 84.9 |
| 4-mer filter | 68.0 | 80.4 | 91.3 | 99.0 | 99.9 | 99.6 | 97.2 | 89.9 | 83.3 | 82.5 |
| 5-mer filter | 66.5 | 77.6 | 88.2 | 98.3 | 99.9 | 99.6 | 97.3 | 89.3 | 80.6 | 79.2 |
| 6-mer filter | 63.1 | 72.1 | 82.3 | 96.1 | 99.9 | 99.6 | 97.4 | 88.5 | 76.2 | 73.3 |
| 7-mer filter | 58.8 | 66.2 | 76.2 | 93.4 | 99.8 | 99.6 | 97.4 | 87.6 | 71.8 | 67.2 |

applications. These metrics were calculated for all values of $D$ between 7 and 11, and for each filtering approach. The results are shown numerically in Table 2 and graphically in Figure 2a.

- We observe that the unfiltered calling approach reaches a significantly higher acceptance rate than the filtering approaches do for the same level of the rejection threshold. For example, when using a rejection threshold of $D = 8$, the unfiltered approach achieves an acceptance rate of 84.9 %, while the 4-mer filtering approach only reaches 80.4 %.
- However, the precision is slightly lower than that of the $k$-mer filtering approaches. For our example of $D = 8$, the no-filter approach achieves a precision of 95.7 %, while the 4-mer filter has 97.2 %.

From our experiments, we conclude that the unfiltered calling approach may be useful in situations where a large acceptance rate is essential and the running time not a limiting problem. However, when aiming for a high precision, a $k$-mer filtering approach and a small value of $D$ is the better choice.

## 5 Read Rejection

In the previous section, the accuracy of Quik has been measured by defining a rejection threshold $D$ and evaluating the acceptance rate and the precision dependent on $D$ in a synthetic experiment. This nicely shows the trade-off between these measures (see Figure 2a). In practice, however, we lack a ground truth, so the precision is unknown. Therefore, it is unclear how to select the appropriate rejection criteria to achieve the desired level of precision. So far, Quik has left the choice of rejection completely to the user. While this gives users ultimate control, it makes Quik less user-friendly. Our aim is to identify a way to determine an appropriate rejection function suitable to be included as the default setting in Quik.

Other tools like Columba [19], Flexiplex [7] and RandomBarcodes [18] also use a rejection threshold $D$ to the best barcode candidate. Columba leaves the choice of threshold completely to the user, while Flexiplex [7] implements a default value that is tailored to the main intended use case of demultiplexing barcodes for Oxford Nanopore. Press [18] derives appropriate thresholds by estimating the distribution of Levenshtein distances between two random strings and statistically computing the minimum distance between a random string and a set of random barcodes, depending on the set's size. This estimation must be repeated for different barcode lengths, alphabet sizes and distance measures. Nevertheless, this aids the user in finding an appropriate threshold for the use cases considered. However, this threshold may be insufficient in certain cases: Consider a read with the same distance $d \leq D$ to two barcode candidates $A$ and $B$. This read will be assigned to either, even though there is no way to determine the correct assignment. This naturally leads to a second rejection

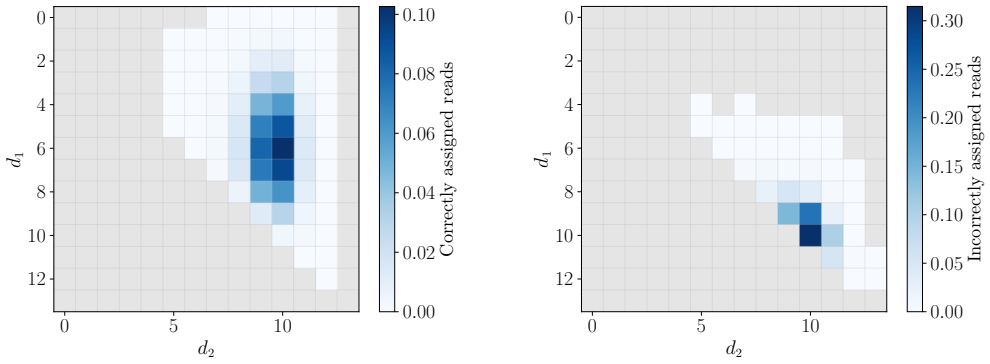



**Figure 3** Read frequency of each pair of distances $(d_1, d_2)$ to the best and second-best barcode candidates of correctly assigned reads (left) and incorrectly assigned reads (right).

strategy: Let $d_1$ be the distance to the best and $d_2$ the distance to the second best barcode candidate. Reject if $d_1$ is larger than $D$ or if the distance difference $d_2 - d_1$ is less than a second threshold $T$. Similar ideas are implemented in Dorado [16], Flexiplex [7], and Barbell [3].

To visualize the motivation for rejection functions, we consider the frequency of simulated reads for each pair $(d_1, d_2)$ after barcode calling. We split the read sets into correctly and incorrectly assigned reads based on the ground truth (Figure 3). The objective is to partition both matrices into accepted and rejected regions in such a way that as many correctly assigned reads as possible are accepted and as many incorrectly assigned reads as possible are rejected. This motivates us to consider linear rejection functions. A read should be rejected if $d_1 + \alpha \cdot d_2 > \beta$ holds for suitable values of the parameters $\alpha$ and $\beta$. See Table 3 for an overview of the rejection functions. The following subsections formalize these ideas, propose approaches for determining optimized parameters and then evaluate and compare the different rejection methods.

## 5.1 Rejection Optimization

First, we need to define the objective of the rejection optimization. Recall Figure 3. Our aim is to divide both matrices into accepted and rejected parts such that, in the accepted region, the number of correctly assigned reads is maximized, while the number of incorrectly assigned reads is minimized. In real experiments, however, we do not know the ground truth and thus cannot use any information on the true assignments to split the read set in the same way. However, we can *simulate* the matrix of incorrectly assigned reads by calling a set of randomly generated strings, as incorrect assignments are basically random. These randomly generated strings do not correspond to any barcode and thus should be rejected. We optimize the parameters of our rejection functions by maximizing the number of accepted reads under the condition that we achieve a pre-defined minimum rate $r_S$ of rejected random strings. By increasing $r_S$, we expect that the rate of incorrectly accepted reads decreases. In our experiments, we set $r_S = 95\%$.

We propose the following approach for the parameter optimization: Given a barcode set and a read set, use Quik to call the reads and aggregate the results to a matrix $R$ in which each entry $R_{d_1,d_2}$ corresponds to the number of reads with distance $d_1$ to the first and $d_2$ to the second best barcode candidate. Next, randomly generate a set $Z$ of strings of the same length as the reads and use Quik to call these strings. Aggregate the results to a matrix $S$ with entries $S_{d_1,d_2}$ which denote the number of strings with distance $d_1$ to the first and $d_2$ to

**Table 3** Rejection functions.

| Name | Rejection Criteria | Optimization Approach |
|---|---|---|
| Simple | $d_1 > D$ | Enumeration for $D \in \{0, ..., \ell\}$ |
| Combined | $d_1 > D \vee d_2 - d_1 < T$ | Enumeration for $D \in \{0, ..., \ell\}, T \in \{0, ..., \ell\}$ |
| Linear | $d_1 + \alpha \cdot d_2 > \beta$ | Integer Linear Program (Gurobi) |
| Upper Bound | $x_{d_1,d_2} = 0$ | Integer Linear Program (Gurobi) |

the second best barcode candidate. Using $R$, we can compute the number of accepted reads for any choice of function parameters. Likewise, we use $S$ to compute the rate of rejected random strings for the same choice of parameters.

To determine an optimal parameter choice for each rejection function, we follow different strategies. The simple and the combined rejection thresholds can be optimized by complete enumeration as the range of possible values of the parameters $D$ and $T$ is small. The linear function is modeled as an integer linear program (ILP) as follows. Let $L$ be the maximum distance of a read or random string to its second closest barcode.

$$\max \sum_{d_1} \sum_{d_2} R_{d_1,d_2} \cdot x_{d_1,d_2} \tag{1}$$

$$\text{s.t.} \sum_{d_1} \sum_{d_2} S_{d_1,d_2} \cdot (1 - x_{d_1,d_2})/|Z| \geq r_S \tag{2}$$

$$x_{d_1,d_2} = 1 \Leftrightarrow d_1 + \alpha \cdot d_2 \leq \beta \qquad \forall d_1, d_2 \in \{0, ..., L\} \tag{3}$$

$$x_{d_1,d_2} \in \{0, 1\} \qquad \forall d_1, d_2 \in \{0, ..., L\} \tag{4}$$

$$\alpha, \beta \in \mathbb{Q} \tag{5}$$

The variables $x_{d_1,d_2}$ indicate whether to accept an assignment with the corresponding distances ($x_{d_1,d_2} = 1$) or to reject it ($x_{d_1,d_2} = 0$). Objective (1) maximizes the number of accepted reads, while Constraint (2) ensures a minimum rate of rejected random strings. Constraints (3) force the indicator variables to correspond to the linear function as a border between accepted and rejected region. The implications are implemented using the Big M method. See Appendix A.1 for the full ILP formulation.

Additionally, we define an upper bound ILP that can freely choose which pairs of distances $(d_1, d_2)$ to reject. It is implemented as a similar model as for the linear rejection function, but omitting Constraints (3) and Variables (5). The solution of the bound ILP could also serve as a special type of rejection function. However, it depends more heavily on the simulated random strings and is therefore more susceptible to statistical error. For example, it may accept reads with very large $d_1$ and $d_2$ because the probability of these values occurring for random strings is low. We expect the other rejection functions to be more robust.

## 5.2 Experimental Setup

As a proof of concept, we performed the following experiment. We re-used the set of $n = 10^6$ random barcodes of length $\ell = 34$. We applied the error model from Uphoff et al. with total error rates of 15%, 20%, 25%, and 30% to generate read sets of $m = 5 \cdot 10^6$ reads of length $\ell = 34$ each. We generated a set of $10^6$ random strings of length $\ell$ for each read set. Each set was called using 4-mer-filtering. The rejection function optimizations were performed using either enumeration or the Python API of Gurobi 13.0.1. In all cases, the running time for the optimization required just fractions of a second and is therefore negligible. After determining optimal rejection functions, we evaluated acceptance rate and precision.

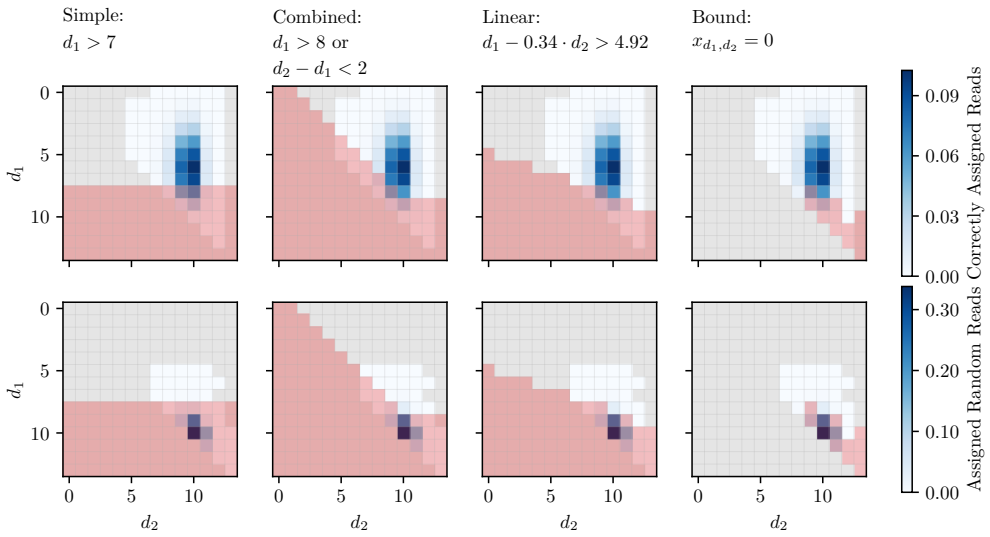



**Figure 4** Overview over the optimal rejection functions for the correctly assigned reads of the set with 20% error rate (top) and the random strings (bottom). The rejected entries are shown in red.

**Table 4** Acceptance rate and precision for each bound type and each read set (given as the error rate in percent) for $r_S = 95\%$.

| | Acceptance rate | | | | Precision | | | |
|---|---|---|---|---|---|---|---|---|
| Set | Simple | Combined | Linear | Bound | Simple | Combined | Linear | Bound |
| 15 | 89.2 | 91.0 | 92.0 | 92.4 | 99.9 | 99.7 | 99.7 | 99.5 |
| 20 | 68.0 | 73.0 | 74.6 | 75.5 | 99.6 | 98.9 | 98.7 | 98.2 |
| 25 | 42.0 | 49.4 | 50.9 | 52.1 | 98.8 | 96.4 | 96.2 | 95.1 |
| 30 | 21.1 | 28.6 | 29.6 | 30.7 | 97.0 | 90.1 | 89.9 | 88.2 |

## 5.3 Results

Figure 4 shows the optimal rejection functions for each rejection type for the read set with 20% error rate. See Appendix A.2 for the corresponding function parameters. The simple rejection threshold is quite strict, rejecting most of the random strings, but also removing a significant proportion of the correctly assigned strings yielding an acceptance rate of 68%. The combined approach increases the acceptance rate to 73% by setting a higher threshold for $d_1$ and additionally restricting the distance difference. This only slightly decreases the precision. The linear rejection function further improves the acceptance rate to 74.6%. It comes close to the acceptance rate of the upper bound ILP, which reaches 75.5%. The resulting pairs of precision and acceptance rate are shown in Figure 2b. The newly proposed rejection functions allow a more fine-grained trade-off. Due to the optimization, we find several new non-dominated pairs of precision and acceptance rate.

Table 4 shows the results for all read sets. The acceptance rate increases from simple over combined to linear, with the linear rejection function being very close to the upper bound. This suggests that more complex rejection functions are more advantageous than simple ones. Conversely, the precision is highest for the simple rejection threshold. The upper bound has the lowest precision, which may be undesirable in some cases. We also see that the results

depend on the error rates used for read generation: For a relatively low rate of 15%, the acceptance rate is at about 90% with a precision of almost 100%, regardless of the rejection type. For the highest tested error rates of 30%, the acceptance rate is only between 20% and 30% with a precision of 88% to 97%. Naturally, it is hard to find the corresponding barcode for such high error rates. In this case, we would expect around 11 errors per read of length 34. Therefore, only a few reads with far fewer than the expected number of errors would have a chance of being assigned correctly. We cannot expect high acceptance rate and precision in these situations. Consequently, 21% acceptance rate at 97% precision with a stricter setting, or 30% acceptance rate at 90% precision seem like reasonable trade-offs. We summarize some key observations:

- The combined and linear rejection functions are superior to the simple one in terms of acceptance rate, and still have an acceptable precision for most use cases.
- The combined rejection thresholds are easy to compute and nearly as good as the linear one, and thus suitable to be included in Quik.
- As the simple one tends to be more strict, the precision is higher.
- For higher error rates, both acceptance rate and precision naturally decrease.
- For very high error rates (30%), a choice must be made whether to prioritize acceptance rate or precision.

This three-step approach of random string generation, calling and optimization is highly flexible and can be applied to any read and barcode set to automatically obtain optimized rejection criteria tailored to the specific data *without* any knowledge of the ground truth. The combined rejection function is particularly easy to compute and close to the upper bound. We therefore consider this to be a valuable addition to Quik.

## 6 Summary

In this paper, we presented several ideas that proved useful to improve the barcode calling software Quik. First, we discussed algorithmic optimization techniques that accelerated the program by a factor of up to 56, while maintaining a high accuracy. Subsequently, we demonstrated that an unfiltered calling approach may give a slightly larger acceptance rate at the cost of a smaller precision and vastly increased running time. Our experiments demonstrate that the $k$-mer filters are to be favored over the unfiltered approach in almost all practical situations. Finally we presented a mechanism to automatically determine rejection criteria optimized to specific barcode and read sets. This greatly increases Quik's usability, as users no longer need to think about an appropriate rejection threshold.

Our improvements are not only instructive from an algorithmic standpoint, but also set new standards for high-precision barcode calling at an unprecedented speed of over 90,000 reads per second. Our tool is freely available at `https://github.com/uni-halle/quik`.

## References

1 Daniel Ashlock and Sheridan K. Houghten. DNA error correcting codes: No crossover. In *2009 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology*, pages 38–45. IEEE, March 2009. doi:10.1109/CIBCB.2009.4925705.

2 James L Banal, Tyson R. Shepherd, Joseph Berleant, Hellen Huang, Miguel Reyes, Cheri M. Ackermann, Paul C. Blainey, and Mark Bathe. Random access DNA memory using boolean search in an archival file storage system. *Nature Materials*, 21:1272–1280, 2021. doi:10.1038/s41563-021-01021-3.

3 Rick Beeloo, Ragnar Groot Koerkamp, Xiu Jia, Marian J. Broekhuizen-Stins, Lieke van IJken, Els M. Broens, Aldert Zomer, and Bas E. Dutilh. Barbell resolves demultiplexing and trimming issues in nanopore data. *bioRxiv*, 2025. doi:10.1101/2025.10.22.683865.

4 Jürgen Behr, Timm Michel, Maya Giridhar, Santra Santhosh, Arya Das, Hamed Sabzalipoor, Tadija Kekić, Etkin Parlar, Igor Ilić, Gisela Ibáñez-Redín, Erika Schaudy, Jory Lietard, Thomas Schletterer, Max Funck, and Mark Somoza. An open-source advanced maskless synthesizer for light-directed chemical synthesis of large nucleic acid libraries and microarrays. *ChemRxiv*, 2024(0222), 2024. doi:10.26434/chemrxiv-2024-j4c90.

5 Tilo Buschmann and Leonid V. Bystrykh. Levenshtein error-correcting barcodes for multiplexed DNA sequencing. *BMC Bioinformatics*, 14(1):272, September 2013. doi:10.1186/1471-2105-14-272.

6 Leonid V. Bystrykh. Generalized DNA Barcode Design Based on Hamming Codes. *PLOS ONE*, 7(5):e36852, May 2012. doi:10.1371/journal.pone.0036852.

7 Oliver Cheng, Min Hao Ling, Changqing Wang, Shuyi Wu, Matthew E Ritchie, Jonathan Göke, Noorul Amin, and Nadia M Davidson. Flexiplex: a versatile demultiplexer and search tool for omics data. *Bioinformatics*, 40(3):btae102, March 2024. doi:10.1093/bioinformatics/btae102.

8 Brant C. Faircloth and Travis C. Glenn. Not All Sequence Tags Are Created Equal: Designing and Validating Sequence Identification Tags Robust to Indels. *PLOS ONE*, 7(8):e42543, 2012. doi:10.1371/journal.pone.0042543.

9 John A. Hawkins, Stephen K. Jones, Ilya J. Finkelstein, and William H. Press. Indel-correcting DNA barcodes for high-throughput sequencing. *Proceedings of the National Academy of Sciences*, 115(27):E6217–E6226, 2018. doi:10.1073/pnas.1802640115.

10 Allon M. Klein, Linas Mazutis, Ilke Akartuna, Naren Tallapragada, Adrian Veres, Victor Li, Leonid Peshkin, Weitz David A, and Marc W. Kirschner. Droplet barcoding for single-cell transcriptomics applied to embryonic stem cells. *Cell*, 161(5):1187–1201, 2015. doi:10.1016/j.cell.2015.04.044.

11 Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

12 Jory Lietard, Adrien Leger, Yaniv Erlich, Norah Sadowski, Winston Timp, and Mark M Somoza. Chemical and photochemical error rates in light-directed synthesis of complex DNA libraries. *Nucleic Acids Research*, 49(12):6687–6701, 2021. doi:10.1093/nar/gkab505.

13 Yang Liu, Mingyu Yang, Yanxiang Deng, Graham Su, Archibald Enninful, Cindy C. Guo, Toma Tebaldi, Di Zhang, Dongjoo Kim, Zhiliang Bai, Eileen Norris, Alisia Pan, Jiatong Li, Yang Xiao, Stephanie Halene, and Rong Fan. High-spatial-resolution multi-omics sequencing via deterministic barcoding in tissue. *Cell*, 183(6):1665–1681.e18, 2020. doi:10.1016/j.cell.2020.10.026.

14 Eli Lyons, Paul Sheridan, Georg Tremmel, Satoru Miyano, and Sumio Sugano. Large-scale dna barcode library generation for biomolecule identification in high-throughput screens. *Scientific reports*, 7(1):1–7, 2017.

15 Timm Michel, Jürgen Behr, Hamed Sabzalipoor, Gisela Ibáñez-Redín, Jory Lietard, Thomas Schletterer, Max Funck, and Mark M. Somoza. Color-corrected and high-contrast catadioptric relay for maskless high-resolution nucleic acid photolithography. *Opt. Express*, 33(8):17068–17084, April 2025. doi:10.1364/OE.557937.

16 Oxford Nanopore Technologies plc. Dorado v1.4.0, 2026. URL: https://github.com/nanoporetech/dorado/.

17 Franco Poma-Soto, Hanne Van Droogenbroeck, Brecht Soulliaert, Maya Giridhar, Jürgen Behr, Hamed Sabzalipoor, Mark Somoza, Pieter Mestdagh, and Jo Vandesompele. Benchmarking DNA barcode decoding strategies under high error rates. *BMC Bioinformatics*, June 2026. doi:10.1186/s12859-026-06540-x.

18 William H Press. Fast trimer statistics facilitate accurate decoding of large random DNA barcode sets even at large sequencing error rates. *PNAS Nexus*, 1(5):pgac252, 2022. doi:10.1093/pnasnexus/pgac252.

19 Luca Renders, Lore Depuydt, Travis Gagie, and Jan Fostier. Columba: fast approximate pattern matching with optimized search schemes. *Bioinformatics*, 41(12):btaf652, December 2025. doi:10.1093/bioinformatics/btaf652.

20 Schüler Steffen, Riko Uphoff, and Antonia Schmidt. Quik. Software, version 2.0., swhId: swh:1:dir:948d8ae0e0b66260e91d201ea0b5f6f732aa4eff (visited on 2026-08-12). URL: https://github.com/uni-halle/quik, doi:10.4230/artifacts.27619.

21 Akshay Tambe and Lior Pachter. Barcode identification for single cell genomics. *BMC Bioinformatics*, 20(1):32, January 2019. doi:10.1186/s12859-019-2612-0.

22 Riko Corwin Uphoff, Steffen Schüler, Ivo Grosse, and Matthias Müller-Hannemann. Fast barcode calling based on k-mer distances. *PNAS Nexus*, 5(2):pgag001, January 2026. doi:10.1093/pnasnexus/pgag001.

23 Johannes Wirth, Nina Huber, Kelvin Yin, Sophie Brood, Simon Chang, Celia P. Martinez-Jiminez, and Matthias Meier. Spatial transcriptomics using multiplexed deterministic barcoding in tissue. *Nature Communications*, 14(1523), 2023. doi:10.1038/s41467-023-37111-w.

24 Eduard Zorita, Pol Cuscó, and Guillaume J. Filion. Starcode: sequence clustering based on all-pairs search. *Bioinformatics*, 31(12):1913–1919, June 2015. doi:10.1093/bioinformatics/btv053.


# A Read Rejection

## A.1 Rejection Optimization

In Section 5, we briefly sketched an ILP for the optimization of the linear rejection function. Using this formulation as is leads to numerical problems when the parameters $\alpha$ and $\beta$ are represented as floating point numbers. Representing $\alpha = \frac{\alpha_n}{\alpha_d}$ and $\beta = \frac{\beta_n}{\beta_d}$ as rational numbers with $\alpha_n, \alpha_d, \beta_n, \beta_d \in \mathbb{Z}$ only partially solves the problem, since with missing bounds on these numbers it remains unclear how to choose the parameter $M$ required to model the implications in Constraints (3) with the Big M method. Therefore, we further restricted these numbers to reach an accuracy of 1% which we consider as sufficient for our purposes. More precisely, we fixed the denominators $\alpha_d$ and $\beta_d$ to 100. The full ILP is given as:

$$\max \sum_{d_1} \sum_{d_2} R_{d_1,d_2} \cdot x_{d_1,d_2} \tag{6}$$

$$\text{s.t.} \sum_{d_1} \sum_{d_2} S_{d_1,d_2} \cdot (1 - x_{d_1,d_2})/|Z| \geq r_S \tag{7}$$

$$d_1 \cdot \alpha_d \cdot \beta_d + \alpha_n \cdot \beta_d \cdot d_2 \leq \beta_n \cdot \alpha_d + M \cdot (1 - x_{d_1,d_2}) \qquad \forall d_1, d_2 \in \{0, ..., L\} \tag{8}$$

$$d_1 \cdot \alpha_d \cdot \beta_d + \alpha_n \cdot \beta_d \cdot d_2 \geq \beta_n \cdot \alpha_d - M \cdot x_{d_1,d_2} + 1 \qquad \forall d_1, d_2 \in \{0, ..., L\} \tag{9}$$

$$\alpha_d = \beta_d = 100 \tag{10}$$

$$-\alpha_d \leq \alpha_n \leq \alpha_d \tag{11}$$

$$-\beta_d \cdot L \leq \beta_n \leq \beta_d \cdot L \tag{12}$$

$$x_{d_1,d_2} \in \{0, 1\} \qquad \forall d_1, d_2 \in \{0, ..., L\} \tag{13}$$

$$\alpha_n, \beta_n \in \mathbb{Z} \tag{14}$$

$$M = L \cdot \alpha_d \cdot \beta_d + L \cdot \beta_d \cdot \alpha_d + L \cdot \beta_d \cdot \alpha_d \tag{15}$$

Inequalities (8) and (9) correspond to Constraints (3) in Section 5. Constraints (10)-(12) are necessary to restrict the variable values for choosing an appropriate value for $M$.

## A.2 Further Results

Table 5 shows the parameters for the optimized rejection functions from Section 5.3. Please note that these parameters highly depend on the settings, particularly the barcode length, set size, alphabet, and distance measure (SL vs. Levenshtein distance). This makes it impossible to derive any universally applicable parameters. The generation of random strings takes into account the distribution of bases and read lengths. Both the read set and the random string set are called with the same settings (the same barcode set and the same distance measure).

To illustrate the dependence of the parameters on the setting, we evaluated a slightly varied experiment: The same barcode set of $10^6$ barcodes and the same error model were used. However, the reads were *not* trimmed to constant length, resulting in reads of *varying* lengths, depending on the number of insertions and deletions in each one. The reads were called with Quik using the Levenshtein distance instead of the SL distance. Table 6 shows the results. The optimal parameters clearly differ between Tables 5 and 6. Thus, the optimized parameters should always be tailored to the specific data set.

**Table 5** Optimized parameters for each bound type and each read set (given as the error rate in percent) for $r_S = 95\%$.

| | Simple | Combined | | Linear | |
|---|---|---|---|---|---|
| Set | D | D | T | $\alpha$ | $\beta$ |
| 15 | 7 | 8 | 2 | −0.34 | 4.92 |
| 20 | 7 | 8 | 2 | −0.34 | 4.92 |
| 25 | 7 | 8 | 2 | −0.11 | 6.99 |
| 30 | 7 | 8 | 2 | −0.34 | 4.92 |

**Table 6** Optimized parameters for each bound type and each read set (given as the error rate in percent) for $r_S = 95\%$ with Levenshtein distance.

| | Simple | Combined | | Linear | |
|---|---|---|---|---|---|
| Set | D | D | T | $\alpha$ | $\beta$ |
| 15 | 8 | 9 | 2 | −0.59 | 2.87 |
| 20 | 8 | 9 | 2 | −0.59 | 2.87 |
| 25 | 8 | 9 | 2 | −0.59 | 2.91 |
| 30 | 8 | 9 | 1 | −0.33 | 5.70 |

# Towards a Unified Exact Solution of Rearrangement Small Parsimony for Natural Genomes

**Leonard Bohnenkämper**
Faculty of Technology and Center for Biotechnology (CeBiTec), Bielefeld University, Germany

**Daria Frolova**
European Bioinformatics Institute, Hinxton, Cambridge, UK
Department of Computational Biology, University of Lausanne, Switzerland

## Abstract

Phylogenetic reconstruction is a fundamental problem in comparative genomics. As a theoretical problem in rearrangement studies, this has been modelled as the Small Parsimony Problem (SPP), in which ancestral genome structures have to be determined minimizing the number of rearrangement events occurring throughout the phylogeny. This problem is of significant interest in microbial and cancer genomics, due to the prevalence and clinical importance of rearrangement events.

Genome structures in this problem are expressed as sequences of *markers*, which are themselves oriented sequence features (such as genes) that abstract from non-structural variations. Recent research has focused on the problem under the natural genomes model, in which arbitrary variations in copy number of markers are allowed. Natural genomes are often studied under the DCJ-indel model, a model which has already been successfully applied to plasmid data. There also exist ILP solutions to a variant of the Small Parsimony Problem under the DCJ-indel model. However, these solutions are limited in their applicability, as they make some critical simplifications for tractability purposes: ancestral marker frequencies and precomputed putative ancestral adjancencies, with their predicted likelihoods, are assumed as input.

This creates multiple problems from both a theoretical and practical perspective. Firstly, this simplification means that not the full state space is searched for a solution, but rather only the subset of genomes with the precomputed putative adjacencies, meaning an optimal solution to the exact SPP is not guaranteed. Secondly, marker frequencies are given externally, without any theoretical guarantees. Thirdly, the method used to precompute adjacencies relies on gene trees, which requires the use of genes as markers, when gene annotation is often unreliable, especially in regions with a lot of rearrangement. Additionally, this restricts the applicability of the approach to sets of genomes that are both divergent and large enough to be able to produce informative gene trees. This is, for example, rarely the case for plasmids, where nucleotide mutations are rarer than rearrangements and genomes are small.

Hence, we revisit the problem to solve the exact SPP by introducing a cost to indel operations, which allows us to compute ranges of marker frequencies and derive theoretical results, that allow us to reduce the solution space that the ILP searches without sacrificing optimality. We show that this makes the problem tractable for the case of small and recently related genomes, first on simulated genomes, and then on a set of pathogenic plasmids which represent a realistic use case for the method.

**2012 ACM Subject Classification** Applied computing → Computational genomics

**Keywords and phrases** Rearrangement, Small Parsimony Problem, Natural Genomes, DCJ-indel

**Digital Object Identifier** 10.4230/LIPIcs.WABI.2026.15

**Related Version** *Preprint*: `https://doi.org/10.64898/2026.06.23.733974`

**Supplementary Material** *Software*: `https://github.com/gi-bielefeld/spp_dcj_exact` [4]
archived at `swh:1:dir:9d2d2213b28d627478afc8a5f365b31b0d0979f1`

**Acknowledgements** LB thanks Roland Wittler for pointing us towards helpful literature concerning Wagner Parsimony as well as Niccolò Gugliette for a helpful discussion.



© Leonard Bohnenkämper and Daria Frolova;
licensed under Creative Commons License CC-BY 4.0





## 1 Introduction

The problem of parsimonious ancestral reconstruction with a given tree is known as the *Small Parsimony Problem (SPP)*. In general, SPP assumes a given tree with known states at the leaves and a cost function penalising changes between states. It then asks to find ancestral states minimising the total cost in the tree. SPP is solved by Fitch's method for nucleotide substitutions under unit cost [18]. However, genomic change is not exclusively characterised by substitutions, and structural changes like gene rearrangements and insertions/deletions (indels) of sequence regions play a significant role in evolution – notable examples include microbial plasmids [24], bacteria [11] and fungi [27], both chromosomal and extrachromosomal DNA in cancer [14, 23], and eukaryotic mitochondrial DNA [2, 32].

To quantify such structural changes various combinatorial rearrangement models have been suggested, in which genomes are treated as sequences or oriented markers, where markers correspond to some sequence features (e.g. genes or synteny blocks). We distinguish here between *singular genomes*, in which each marker occurs at most once and *natural genomes*, for which no restrictions pertaining to copy number is made. In the context of rearrangement models, SPP then asks to reconstruct ancestral marker orders which minimise the total number of rearrangements under a given model.

As a fundamental reconstruction problem, the small parsimony problem has been studied under various rearrangement models. Notably, even for singular genomes, the problem is already NP-hard for only three leaves (median of 3 problem) under complex models, such as the Double-Cut-And-Join model [29] or the Reversal model [9]. In fact, the Problem is only known to be polynomially solvable for the simple Single-Cut-or-Join model [17], which however is not able to handle arbitrary natural genomes.

One rearrangement model that has been successfully extended to handle natural genomes is the *Double-Cut-And-Join and indel (DCJ-indel) model* [8, 5, 3]. The rearrangement operations in this model are Double-Cut-And-Join (DCJ) – a general rearrangement operation able to mimic reversals, translocations and transpositions – as well as insertions and deletions of one or multiple markers at once. As determining the minimum number of DCJs and indels between a pair of natural genomes is already NP-hard [5], the solutions to this problem [5, 3] are based on an Integer Linear Program (ILP) [26].

The first attempts at a solution to DCJ-indel SPP for natural genomes were undertaken by Doerr and Chauve [13] and Bohnenkämper *et al* [7]. However, owing to the computational complexity of the problem, these solutions rely on several simplifications. Firstly, the copy number of each marker at ancestral nodes is assumed as input. Secondly, a collection of putative ancestral adjacencies is precomputed, and the ILP searches only within the space of these previously inferred adjacencies, rather than the full space of adjacencies. Thirdly, these precomputed adjacencies are given a weight, and this weight is incorporated into the objective function of the ILP. Although these simplifications help make the problem tractable, it introduces several critical limitations. Since only a part of the space of adjacencies is searched, it does not truly solve the SPP, and an optimal solution on the full space is not guaranteed. Additionally, the current method used to precompute ancestral adjacencies and their weights relies on gene trees [15]. Gene annotation is often unreliable, especially in regions with structural variation (e.g. even two identical genomes can be annotated differently due to genes being missed), making genes difficult to use as markers in practice [19, 30]. Moreover, grouping genes into homologous groups is also error prone and for recently related or very small genomes there is little SNP information to build gene trees from, making them insufficiently informative for the problem, and therefore limiting its applicability to between-species comparisons.

Naively, one can still achieve a solution to the exact DCJ-indel SPP, and avoid reliance on gene trees, by choosing weights of zero and inputting the entire space of adjacencies. However, this requires knowledge of the correct copy number for each internal node, for which so far no theoretical result exists. Moreover, in practice, such large input data cannot be handled by ILP solvers efficiently, as we demonstrate in Section 4.

In this work, we make the first step to resolve these problems, establishing a way to infer copy numbers using a modified version of the DCJ-indel distance and using properties of this distance measure – notably a restored triangle inequality – to restrict the solution space from all adjacencies to a set of candidates that guarantees at least one optimal solution remains.

The remainder of this manuscript is organised as follows. In Section 2, we give basic definitions and detail previous work. In Section 3.1, we discuss how to parsimoniously infer marker frequencies using a Fitch-like approach. In Sections 3.2 and 3.3, we introduce a way to restrict the solution space, such that at least one optimal solution under the DCJ-indel model is preserved. In Section 3.4, we introduce a way to use strategies like weighting adjacencies as a runtime heuristic as opposed to a correctness heuristic in an ILP context. We evaluate our solution in Section 4 and discuss our findings in Section 5.

## 2 Background

In rearrangement studies, one often abstracts from the concrete sequence content to the higher level of synteny blocks or *markers*. Formally, a marker $m$ is a symbol from a (large) alphabet. To represent strandedness, markers are assigned an *orientation*, i.e. $+m$ or $-m$. We then speak of an *oriented marker*. A *chromosome* is then modeled as a circular or linear string over the alphabet of oriented markers and a *genome* as a collection of chromosomes.

In our notation, we write a circular string in round and a linear string in square brackets. For example, $E = \{[+1 - 2 - 1], (+3), (+3)\}$ is a genome with a linear and two circular chromosomes. Note that we do not restrict the number of occurrences of a chromosome or marker in our definitions. Also note that inversions of chromosomes are considered equivalent, i.e. $[+1 + 2 + 3] = [-3 - 2 - 1]$ as well as rotations of circular chromosomes, i.e. $(+1 - 2) = (-2 + 1)$.

We define the *universe of natural genomes* $\mathcal{U}$ as the set of all possible genomes over a given marker alphabet. Observe that $\mathcal{U}$ is infinite even though the underlying marker alphabet considered is finite.

The *Small Parsimony Problem* is a reconstruction problem, in which ancestral states are reconstructed for a known tree and known extant genomes in such a manner that the minimum number of evolutionary events is needed to "explain" the resulting phylogeny.

Formally, a binary tree $T$ is a connected acyclic graph, such that each node has at most three neighbors. A node is called a *leaf* if it has degree 1, otherwise it is an *internal node*. Sometimes it is helpful to examine only a part of the tree by considering a *subtree* of $T$. To that end, removing some edge $(u, r) \in E(T)$ splits the tree into two components. Formally, the *subtree* with *root* $r$ is then the component containing $r$. In the Small Parsimony Problem, each leaf represents an extant genome whereas each internal node represents the most recent common ancestor of all leaves in one of the subtrees of which it is the root.

Given a rearrangement distance measure $d$, the Small Parsimony Problem in the context of natural genomes can then be simply defined as follows.

▶ **Problem 1** (Small Parsimony Problem for Natural Genomes)**.** *Given a binary tree $T$ with edges $E(T)$ and genome labels $L_u \in \mathcal{U}$ for each leaf $u$, find labels $L_v \in \mathcal{U}$ for each internal node $v$, such that*

$$\sum_{(v',u')\in E(T)} d(L_{v'}, L_{u'})$$

*is* minimised *under distance $d$.*

## 2.1 Previous Work

In this subsection, we summarise previous work, which is necessary to derive the results of our contribution. We begin by discussing the DCJ-indel distance in Section 2.1.1 and the previous approach by Doerr and Chauve to solve a restricted version of SPP in Section 2.1.2.

### 2.1.1 DCJ-indel distance for natural genomes

Here we summarise previous results on an important part of the distance model $d$ that we will later use in Problem 1. Rearrangement distances between two genomes $A, B \in \mathcal{U}$ are often defined as the minimum number of rearrangement operations under a chosen model needed to transform $A$ into $B$. The problem of determining this minimum number of rearrangements between two genomes is known as the *distance problem*. To that end, it is often useful to disambiguate between different occurrences of the same marker. We thus presume that there is some arbitrary labelling $A^l$ for each genome $A$ that adds a unique sub-indentifier $i$ to each occurrence of a marker $m$, obtaining $m_i$. We call $m_i$ a *marker occurrence* and $A^l$ a *labelled genome*. For example, $E^l = \{[+1_1 - 2_1 - 1_2], (+3_1), (+3_2)\}$ would be one of multiple labellings for the genome $E$ introduced previously. From now on, we presume each genome is a labelled genome.

Each (labelled) genome can be equivalently described by its *adjacencies*. For each marker occurrence $m_i$ in $A$, we denote its beginning by $m_i^t$ (tail) and its end by $m_i^h$ (head), i.e. $+m_i = (m_i^t, m_i^h)$ and $-m_i = (m_i^h, m_i^t)$. We call $m_i^h, m_i^t$ *extremities*. An *adjacency* is then a pair of extremities $\{m_i^x, n_j^y\}$, $x, y \in \{t, h\}$, such that the corresponding marker occurrences $m_i$ and $n_j$ are neighboring on a chromosome and $m_i^x$ and $n_j^y$ are adjacent. For example, in $E^l$ we have the adjacencies $\{1_1^h, 2_1^h\}, \{2_1^t, 1_2^h\}, \{3_1^h, 3_1^t\}, \{3_2^h, 3_2^t\}$. In linear chromosomes, the first extremity of the first marker and second extremity of the last marker are not adjacent to any other extremity. For mathematical convenience we thus define a *telomeric adjacency* $\{o_k^z, \varnothing(o_k^z)\}$ for such an extremity $o_k^z$ for a unique artificial extremity $\varnothing(o_k^z)$. $\varnothing(o_k^z)$ is called a *pseudo-cap*. For example, in $E^l$ we have the telomeric adjacencies $\{1_1^t, \varnothing(1_1^t)\}$ and $\{1_2^t, \varnothing(1_2^t)\}$. When the concrete extremities are irrelevant, we use the shorthand notation $ab = \{m_i^x, n_i^y\}$ to denote the adjacency as an arbitrary pair of extremities $a$ and $b$. Note that the order of $a$ and $b$ is still arbitrary, i.e. $ab = ba$.

A relatively simple but powerful rearrangement model is the DCJ-indel model [8]. It was first introduced on *singular genomes*, a finite subset $S \subset \mathcal{U}$ of all natural genomes. In a singular genome each marker may occur at most once. To this end, we define the *copy number* $\phi(A, m)$ of marker $m$ as the number of times marker $m$ occurs in genome $A$ as some occurrence $m_i$. For example, earlier in $E^l$ we have $\phi(E^l, 1) = \phi(E^l, 3) = 2, \phi(E^l, 2) = 1$. We can then define the set of singular genomes as $S \coloneqq \{A \in \mathcal{U} \mid \forall m\, \phi(A, m) \leq 1\}$. Moreover, we say a marker $m$ with $\phi(A, m) = 1$ and its extremities are *singular* in $A$.

Within the DCJ-indel model, the following operations are permitted:

- A *Double-Cut-And-Join (DCJ)* transforms the adjacencies of a genome in one of the following ways:
  - $ab, cd \to ac, bd$
  - $a\varnothing(a), b\varnothing(b) \to ab$
  - $ab, c\varnothing(c) \to ac, b\varnothing(b)$
  - $ab \to a\varnothing(a), b\varnothing(b)$
- An *insertion or deletion* (indel), in which a segment of one or multiple markers is inserted or deleted at any position in the genome.

In order to prevent "free lunch" scenarios, in which chromosomes are simply deleted as a whole, the original formulation in [8] allows only to insert or delete non-shared markers between the two genomes. The DCJ-indel distance $d_{\mathrm{DCJ}}^{\mathrm{id}}(A, B)$ for two singular genomes $A, B \in S$ is then the minimum number of DCJ and indel operations needed to transform one genome into the other.

This distance can be calculated in linear time using the *relational diagram*. Given two singular genomes $A$ and $B$, the *Relational Diagram (RD)* $R(A, B)$ is a graph $(V, E)$. Its set of vertices is $V \coloneqq V(A) \cup V(B)$, where $V(A)$ has a vertex for each extremity of each marker occurrence of genome $A$, and $V(B)$ has a vertex for each extremity of each marker occurrence of genome $B$. For any adjacency $ab$ in $A$, an *adjacency edge* is added between vertices $a$ and $b$, and similarly for any adjacency in $B$. Additionally, for every shared marker $m$, any $m_i^h$ in $V(A)$ is connected to any $m_j^h$ in $V(B)$ by an *extremity edge*, and similarly any $m_i^t$ in $V(A)$ to any $m_j^t$ in $V(B)$. An example for a RD is given in Figure 1.

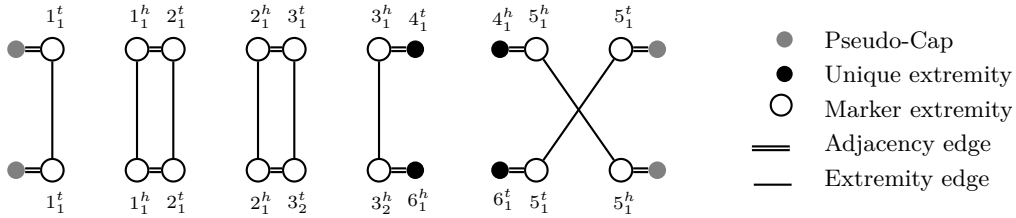



**Figure 1** Relational Diagram (RD) for singular genomes $A = \{[+1_1 + 2_1 + 3_1 + 4_1 - 5_1]\}$ and $B = \{[+1_1 + 2_1 + 3_1 - 6_1 + 5_1]\}$.

Notably, since each marker occurs at most once in each genome, the RD consists only of simple cycles and paths. Paths can either end in a pseudo-cap, in which case we denote the path end by the genome name to which the pseudo-cap belongs in capital letters ($A$ or $B$) or in an extremity unique to one of the genomes, in which case we denote the path end by the genome name in lowercase letters ($a$ or $b$). All combinations of these path ends yield 10 different path types, the counts of which in a given Relational Diagram we denote accordingly by $p_{AA}, p_{AB}, p_{BB}, p_{Aa}, p_{Ab}, p_{Ba}, p_{Bb}, p_{aa}, p_{ab}, p_{bb}$. We write the number of cycles as $c$. For example, in Figure 1, we have $c = 2, p_{AB} = 1, p_{ab} = 1, p_{Ab} = 1, p_{Ba} = 1$ with all other counts being 0.

*Circular singletons* are structures that are not represented in this graph (see [3] for details), but still necessary to calculate the DCJ-indel distance. A circular singleton is a circular chromosome consisting only of markers unique to one genome. We denote the number of circular singletons by $s$. In [3], the following result is derived.

▶ **Theorem 1** (adapted from [3]). *For two singular genomes $A, B$, the DCJ-indel distance is*

$$d_{DCJ}^{id}(A, B) = n - c + \left\lceil \frac{p_{ab} + \max(p_{Aa}, p_{Ba}) + \max(p_{Ab}, p_{Bb}) - p_{AB}}{2} \right\rceil + s$$

*with $n$ the number of* shared markers.

The distance problem of natural genomes can be reduced to the distance problem of singular genomes by employing a *matching*, which informs for a marker $m$ which occurrence $m_i$ in $A$ should be considered the same as $m_j$ in $B$. A matching between genomes $A$ and $B$ is defined as a set of pairs $M = \{(m_i, m_j) \mid m \text{ shared marker}, m_i \text{ occurrence in } A, m_j \text{ occurrence in } B\}$, such that each marker occurrence appears at most once in $M$.

Given a matching for each marker in $A$ and $B$, we can compare them as singular genomes. For example, given $(m_i, m_j) \in M$ can then rewrite both $m_i$ in $A$ and $m_j$ in $B$ to the same unique character $u^{ij}$, and rewrite any occurrence not in $M$ to a character that does not occur in the other genome. We call the resulting genomes $A^M, B^M$ the *singularisations* of $A$ and $B$. The distance between two natural genomes can then be defined as the minimum distance between any two singularisations obtained by a matching.

However, not all matchings are meaningful. For example, choosing the empty matching would lead to all chromosomes being inserted or deleted, which under the DCJ-indel distance may actually be optimal, since it requires only one (indel) operation per chromosome. One way to resolve this issue is to require the matching to be *maximal*, i.e. such that it cannot be extended. In this case, this is equivalent to the matching having maximum cardinality. Formally, we write $\mathbb{M}(A, B)$ as the set of all possible matchings between two genomes $A, B$ and $\overline{\mathbb{M}}(A, B) \coloneqq \{M \in \mathbb{M}(A, B) \mid \nexists M' \in \mathbb{M}(A, B), |M'| > |M|\}$ as the set of *maximal matchings*.

In [5, 13, 7], the distance measure $\overline{d^{\text{id}}_{\text{DCJ}}}$ for two natural genomes $A, B$ is defined as the minimum DCJ-indel distance between any singularisations obtained by a maximal matching:

$$\overline{d^{\text{id}}_{\text{DCJ}}}(A, B) = \min_{M \in \overline{\mathbb{M}}(A,B)} d^{\text{id}}_{\text{DCJ}}(A^M, B^M). \tag{1}$$

We call this distance model the *restricted* DCJ-indel distance.

### 2.1.2 Linearising degenerate genomes

The first attempt to solve the Small Parsimony Problem under the DCJ-indel model for natural genomes was undertaken by Doerr and Chauve in 2021 [13]. In their formulation, the authors restrict Problem 1 in two critical ways: (I) It is assumed that the copy number $\phi(L_v, m)$ of the genome label $L_v \in \mathcal{U}$ internal node $v$ is given for each marker $m$. This restricts the infinite universe $\mathcal{U}$ to a finite subset of candidate genomes. (II) It is assumed that even within that restricted subset, only some adjacencies are permitted, further restricting the space of possibilities.

The first assumption about marker copy numbers was relaxed in [7], allowing for a range of given copy numbers $l(v, m) \leq \phi(L_v, m) \leq h(v, m)$ for the genome label $L_v$ at internal node $v$. Nonetheless, this solution still relies on external heuristics to derive copy number counts and candidate adjacencies, which inform the finite candidate set that is explored. The advantage of this formulation is that the solution space with given input copy numbers and adjacencies can be encoded as a *degenerate genome*.

▶ **Definition 2** (adapted from [13]). *A* degenerate genome *is a set of adjacencies $D$ that satisfies the following conditions: (i) $\forall g_i^t \in \bigcup D$, there exists also extremity $g_i^h \in \bigcup A$ and vice versa, and (ii) each telomeric extremity $\varnothing(g_i^x)$ is used only in the adjacency $\{g_i^x, \varnothing(g_i^x)\}$.*

The solution space spanned by a degenerate genome $D$ is then each genome that has only adjacencies from $D$. Such a genome is called a *linearisation of $D$*. Formally, a *linearisation* of degenerate genome $D$ is a genome $A$, such that each adjacency $\{m_i^x n_j^y\}$ of $A$ is part of $D$, i.e. $\{m_i^x n_j^y\} \in D$.

In the original formulation [13], each extremity in $D$ must be used in the linearisation $A$. However, since the ILP solution in [7] allows for variant copy numbers, we choose a slight reformulation that explicitly encodes the upper and lower bounds for each marker. The variant of SPP solved in [7] can then be formulated as follows.

▶ **Problem 2** (adapted from [7]; Weighted Small Parsimony Linearisation Problem)**.** *Given a binary tree $T$ with edges $E(T)$, genome labels $L_u \in \mathcal{U}$ for each leaf $u$, degenerate genomes $D_v$ for each internal node $v$, a weighting function $w_v$ for adjacencies of each degenerate genome $D_v$, lower and higher bounds $l(v,m), h(v,m)$ for each marker $m$ at each internal node $v$ and a parameter $\alpha \in [0,1]$, find a linearisation $L_v$ for each degenerate genome $D_v$, such that $l(v,m) \leq \phi(L_v,m) \leq h(v,m)$, minimising*

$$\sum_{(u,v)\in E(T)} \left( \alpha\, \overline{d^{id}_{DCJ}}(L_u, L_v) \;+\; (\alpha - 1) \left( \sum_{\substack{ab \\ adj.\ of\ L_v}} w_v(ab) \;+\; \sum_{\substack{ab \\ adj.\ of\ L_u}} w_u(ab) \right) \right).$$

Observe that what is optimised in this problem formulation is not the restricted DCJ-indel distance, but, depending on $\alpha$, also a weight bias $w_v$ toward the input adjacencies at each node $v$. These weights are part of the input and result from confidence scores of the adjacency prediction for ancestral genomes. The consequence of this formulation is that the set of degenerate genomes and weights associated with each of their adjacencies are assumed to be given, and must be precomputed before solving Problem 2. Since these are derived externally and without any theoretical guarantees, this formulation does not solve the SPP for natural genomes as formulated in Problem 1.

Note however, that if one could guarantee that the optimal solution is encoded in the degenerate genomes $D_v$ and set $\alpha = 1$, Problem 2 could be used to solve to Problem 1. In the following, we will develop the theoretical background that allows us to do exactly this. We will show how to restrict the copy number of each marker to guarantee all optimal solutions are still represented and derive results that help to restrict the space of possible adjacencies while guaranteeing at least one optimal solution can be found.

## 2.2 Problem Definition

Besides restricting the infinite solution space $\mathcal{U}$ to the finite solution space expressible by degenerate genomes, the reason why marker copy numbers $\phi$ are assumed to be given in [13] and [5], is that using $d = \overline{d^{\mathrm{id}}_{\mathrm{DCJ}}}$ in Problem 1 would inherently cause problems due to some adverse properties of the restricted DCJ-indel distance. Importantly, this distance is not a metric, because it violates the triangle inequality [8].

While this distance superficially resembles a transformation distance and therefore would automatically fulfil the triangle inequality, it is not, because not all intermediate genomes are permitted when transforming one genome into the other.

The most striking example for this is choosing any two unichromosomal genomes $A = \{[S_A]\}, B = \{[S_B]\}$ for any oriented strings $S_A$ and $S_B$ over the marker alphabet that have distance $\overline{d^{\mathrm{id}}_{\mathrm{DCJ}}}(A,B) > 2$. If we now regard the empty genome $C = \{\}$, we observe that $\overline{d^{\mathrm{id}}_{\mathrm{DCJ}}}(A,C) + \overline{d^{\mathrm{id}}_{\mathrm{DCJ}}}(C,B) = 1 + 1 < \overline{d^{\mathrm{id}}_{\mathrm{DCJ}}}(A,B)$ as only a single segmental deletion or insertion is necessary to transform $A$ into $C$ or $C$ into $B$, but when transforming $A$ into $B$, $C$ is not permitted as an intermediate genome. The empty genome is thus almost always among the genomes with the lowest total restricted DCJ-indel distance to any set of genomes. Thus, solving Problem 1 simply setting $d = \overline{d^{\mathrm{id}}_{\mathrm{DCJ}}}$ will result with high likelihood in empty or near-empty ancestral genomes.

However, instead of restricting marker multiplicities externally, we opt to instead restore the triangle inequality for the DCJ-indel distance by permitting all intermediate genomes, but penalising excessive gains and losses. To that end, we adapt the solution suggested by Braga, Willing and Stoye in the original publication of the DCJ-indel model [8]. There the authors propose to use a surcharge $\delta$ for each inserted or deleted marker between two singular genomes. Essentially, an indel, instead of having fixed cost 1, instead has cost $1 + \delta l$ where $l$ is the length of the inserted or deleted segment. For singular genomes, this cost can then be calculated after determining the default DCJ-indel distance by simply examining the non-shared markers. Defining the non-shared markers as $A \triangle B \coloneqq \{m \mid \phi(A, m) = 1, \phi(B, m) = 0\} \cup \{m \mid \phi(A, m) = 0, \phi(B, m) = 1\}$, this distance is then $d^s(A, B) = d^{\text{id}}_{\text{DCJ}}(A, B) + \delta|A \triangle B|$.

Note that this result is restricted to singular genomes $S$. In order to apply it to all natural genomes $\mathcal{U}$, we can generalise as follows.

$$d(A, B) \coloneqq \min_{M \in \mathbb{M}(A,B)} d^{\text{id}}_{\text{DCJ}}(A^M, B^M) + \delta|A^M \triangle B^M| \tag{2}$$

Importantly, we allow any matching $M \in \mathbb{M}(A, B)$ here, not just maximal matchings, permitting any intermediate genome. Observe that then $d$ is equivalent to the minimum cost of transforming $A$ into $B$ when each DCJ has cost 1 and each indel of length $l$ costs $1 + \delta l$, but with no restrictions on which markers can be inserted or deleted – even shared markers can be part of indel operations. Thus, $d$ is a transformation distance. We then arrive at the following.

▶ **Theorem 3.** *$d$ is a metric on $\mathcal{U}$ for any $\delta \in \mathbb{R}, \delta \geq 0$.*

**Proof.** Identity and symmetry follow directly from the identity and symmetry of the DCJ-indel distance and identity and symmetry of $|A^M \triangle B^M|$. What remains to be shown is the triangle inequality, which follows directly from $d$ being a transformation distance. Let thus $A, B, C \in \mathcal{U}$. Observe that if transforming $A$ into $C$ has cost $d(A, C)$ and $C$ into $B$ has cost $d(C, B)$, then there is a way to transform $A$ into $B$ with cost $d(A, C) + d(C, B)$. Thus, $d(A, C) + d(C, B) \geq d(A, B)$. ◀

In principle, we can thus choose any $\delta$ for our formulation, but choosing a small $\delta$ will lead to the same problems as the pure DCJ-indel distance. In previous work on natural genomes, these problems are circumvented by allowing only maximal matchings [5, 13, 7].

Note maximising the matching size $|M|$ is equivalent to minimising $|A^M \triangle B^M|$. In the context of our distance model $d$, this is then equivalent to regarding $\delta \to \infty$, so that any difference in $\delta|A^M \triangle B^M|$ is greater than any difference in $d^{\text{id}}_{\text{DCJ}}$, enforcing as many markers as possible are matched. This means for our formulation we can first minimise $\delta|A^M \triangle B^M|$ and then minimise $d^{\text{id}}_{\text{DCJ}}$ under the maximal matching model.

## 3 Methods

Our method rests on reducing Problem 1 to Problem 2, that is, we find degenerate genomes that guarantee to encode the optimal linearisations from the universe $\mathcal{U}$. To that end, knowing the copy number for each marker is essential. Since we presume $\delta|A^M \triangle B^M|$ is the dominant term in $d$ and can therefore be optimised in advance, we first derive marker copy numbers along the tree based on this cost in Section 3.1. Once we know the copy number bounds $l(v, m), h(v, m)$ for each marker $m$ at each internal node $v$, we can then create the degenerate genome $D_v$ consisting of *all possible adjacencies* for the given extremities. We call

such a genome *fully degenerate*. Concretely, the extremities given the copy number bounds are $X = \{m_i^t \mid \forall m, 1 \leq i \leq h(v,m)\} \cup \{m_i^h \mid \forall m, 1 \leq i \leq h(v,m)\}$, and the fully degenerate genome is $D_v = \{\{x,y\} \mid x \in X, y \in X\} \cup \{(x, \varnothing(x)) \mid x \in X\}$.

In principle, this then allows us to solve SPP by solving a variant of Problem 2. We do this by setting $\alpha = 1$ and enforcing in the ILP of [7] that the total cost contributed by $\sum_{(u,v)\in E(T)} \delta |L_u^M \triangle L_v^M|$ in the tree is exactly the minimum calculated beforehand.

However, in practice, it makes sense to further restrict the space of possible adjacencies. To accomplish this in a manner that guarantees at least one optimal solution remains, further theoretical results are necessary. We derive these results in Section 3.2.

Moreover, when we consider the full space of adjacencies, this opens up a way to restrict the space of possible matchings, a technique that could not be used in previous solutions to the problem. We describe the relevant theoretical results in Section 3.3.

Finally, we develop in Section 3.4 how we can use aspects of correctness heuristics like the weights used in Problem 2 as part of our runtime heuristic to obtain good solutions early in the solving process.

## 3.1 Finding Optimal Marker Frequencies

Since under our distance function a cost of $\delta$ is needed for each marker that is not matched to another, changes in copy number of marker $m$ between two linearisations $L_u, L_v$ that are adjacent in the phylogeny ($(u,v) \in T$) imply a cost of $\delta|\phi(L_u, m) - \phi(L_v, m)|$. We can thus infer marker copy numbers under this model, minimising this total cost across the phylogeny. Formally, this is equivalent to a character-state small parsimony problem, where each marker is a character and states are copy numbers, drawn from the alphabet $\mathbb{N}_0$ and the cost function is $c(x,y) = \delta|x-y|$. Since any cost in this model is scaled by $\delta$, we can instead equivalently consider the cost function $c'(x,y) = |x-y|$.

This is then a Wagner Parsimony Problem, for which ancestral copy numbers are easily computed using the Fitch-like Farris optimisation [16]. An efficient, polynomial algorithm to construct all possible character state ranges is given by Swofford and Maddison in [28].

## 3.2 Optimally Filtering Adjacencies

With the maximum copy number per marker known, we can now build a fully degenerate genome for each internal node that is guaranteed to have all optimal genome labels from $\mathcal{U}$ for that node as linearisations. However, this solution space is much larger than needed for many problem instances. Particularly, since we are interested in phylogenies with a recent history, distances in subtrees close to the leaves are expected to be small. The following lemma allows us to use this fact to restrict the solution space for adjacencies in these cases. To that end, notice that the adjacencies of a genome $A$ are themselves a degenerate genome that permits exactly one linearisation, namely $A$. Therefore, any Lemma that applies to degenerate genomes also applies to genomes in general.

▶ **Lemma 4.** *Given three (degenerate) genomes $D_v, D_u, D_w$, at distinct nodes $u, v, w$, such that $(u,w), (v,w) \in E(T)$ and the maximum distance between any linearisations of $D_v$ and $D_u$ is $d_{u,v}^{\max}$ and $D_u \subset D_w$. For the optimal linearisations $L_w^*$ of $D_w$ and $L_v^*$ of $D_v$ holds $d(L_v^*, L_w^*) \leq d_{u,v}^{\max}$.*

**Proof.** For the sake of contradiction assume $d(L_v^*, L_w^*) > d_{u,v}^{\max}$. Let $(x,w) \in E(T)$ with $x \neq u, x \neq v$, i.e. $x$ is the third node adjacent to $w$ in the tree with degenerate genome $D_x$ and optimal linearisation $L_x^*$. The total cost contributed to the tree at node $w$ by $L_w^*$ is then

$d(L_v^*, L_w^*) + d(L_u^*, L_w^*) + d(L_w^*, L_x^*) > d_{u,v}^{\max} + d(L_u^*, L_w^*) + d(L_w^*, L_x^*) \geq d_{u,v}^{\max} + d(L_u^*, L_x^*) = d(L_u^*, L_u^*) + d(L_u^*, L_v^*) + d(L_u^*, L_x^*)$. Therefore, replacing $L_w^*$ with $L_u^*$ as a linearisation of $D_w$ would yield a smaller total cost. $L_w^*$ would thus not be optimal, a contradiction. ◀

Most importantly, we can use this result for two adjacent leaves ("cherries") that have a distance of at most 1.

▶ **Corollary 5.** *For two leaves $u, v$ incident at internal node $w$, $(u, w), (v, w) \in E(T)$, such that $d(L_u, L_v) \leq 1$. Then the optimal linearisation $L_w^*$ of $D_w$ is $L_u$ or $L_v$.*

In fact, we can iterate this argument: Since $L_w^*$ is then either $L_u$ or $L_v$, we can restrict the degenerate genome $D_w$ to only permit $L_u$ or $L_v$ as linearisations. If we then consider two other distinct nodes $x, w'$ with $(x, w), (x, w') \in E(T)$, such that all linearisations of $D_w, D_{w'}$ have distance of at most one, we can restrict the candidate set $D_x$ as well. As long as the distance between genomes is at most 1, we can remove entire subtrees from the problem without sacrificing optimality.

▶ **Corollary 6.** *The optimal linearisation $L_r^*$ of $D_r$ of any subtree $E'$ with root $r$ and leaves $L'$, such that for all $u, v \in L'$ $d(L_v, L_u) \leq 1$ is a genome from $\{L_v \mid v \in L\}$.*

We can then remove all nodes of $T'$ except $r$ from the tree. Moreover, the optimal linearisation for all other nodes can be determined using the Fitch algorithm: In a bottom up pass, linearisation candidates at each internal node are determined and after the optimal linearisation $L_r^*$ is found, a top down pass decides which linearisation candidate is used.

These results help to keep the solution space small when distances are small. However, even when distances are larger, individual adjacencies often stay preserved within a subtree. The next observation allows us to exploit this fact.

▶ **Lemma 7.** *Given (degenerate) genomes $D_u, D_v, D_w, D_x$ for distinct nodes $u, v, w, x$, such that $(u, w), (v, w), (w, x) \in E$, and such that any optimal linearisations $L_u, L_v$ of $D_u$ and $D_v$ both contain adjacency $ab$ where $a, b$ are singular extremities and $D_w$ is fully degenerate, but any optimal linearisation contains $a, b$ as singular extremities. Then for any optimal linearisation $L_w^*$ of $D_w$, there is a linearisation $L'_w$ that contains $ab$ and is co-optimal.*

**Proof.** Regard optimal linearisations $L_u^*, L_v^*, L_w^*, L_x^*$ of $D_u, D_w, D_w, D_x$. Assume $L_w^*$ does not contain $ab$, but instead $aa'$ and $bb'$ with $a' \neq b, b' \neq a$. Regard $L'_w$, which differs from $L_w^*$ by instead containing adjacencies $ab$ and $a'b'$, all other adjacencies being identical to $L_w^*$. Observe that $d(L'_w, L_x^*) \leq d(L_w^*, L_x^*) + 1$, since $L'_w$ differs from $L_w^*$ by a single DCJ. Moreover, the components in the Relational diagrams between $L'_w, L_u^*$ and $L'_w, L_v^*$ differ from those of $L_w^*, L_u^*$ and $L_w^*, L_v^*$ by only an extra cycle (see Figure 2). This means the term $c$ (see Theorem 1) is increased by one. Since no other component types are changed, all terms based on the graph remain the same. The number of common markers $n$ remains also the same. The only term that could have changed otherwise is $s$, the number of circular singletons. However observe that if $a'$ and $b'$ are part of a circular singletons, it means their markers incur a cost of $\delta$ upon being deleted when transforming $L_w^*$ into $L_u^*$ or $L_v^*$. Since $\delta|A^M \triangle B^M|$ is the dominant term in $d$, $L_w^*$ could not have been optimal if they are circular singletons with both $L_u^*$ and $L_v^*$. Let thus w.l.o.g. $a'b'$ not be a circular singleton for $L'_w, L_u^*$. Then, $d(L'_w, L_u^*) = d(L_w^*, L_u^*) - 1$ and $d(L'_w, L_v^*) \leq d(L'_w, L_v^*)$. With $d(L'_w, L_x^*) \leq d(L_w^*, L_x^*) + 1$, $L'_w$ is thus at least co-optimal to $L_w^*$. ◀

Again, this restriction can be propagated through the tree. As long as an adjacency consists of singular extremities and is preserved in the subtree, no alternatives have to be considered. Once duplicates are involved, the solution space is better constrained on the matchings than the adjacencies as we delineate in the next subsection.

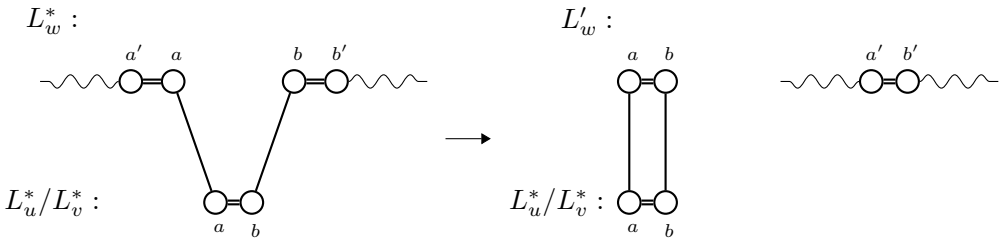



**Figure 2** Difference for the Relational Diagrams of $L_w^*$ (top left) and $L'_w$ (top right) with $L_v^*$ or $L_u^*$ (both bottom). Squiggled lines represent arbitrary simple paths. $L'_w$ has an extra cycle and otherwise the same graph component types. Particularly, the type of component that $a', b'$ are in is unchanged.

## 3.3 Optimally restricting the Matching Space

Note that with duplicates, we are not able to meaningfully restrict the adjacency space unless distances are small (Corollary 6). However, because we are using the full adjacency space, we are able to restrict the number of different matchings. To see how this is possible observe the following pair of labeled genomes: $E_2 = \{[+1_1 - 2_1 + 2_2]\}, E_3 = \{[+1_1 - 2_2 + 2_1]\}$. Both are distinct linearisations of the same fully degenerate genome. However, both linearisations represent the same structure – the distance to any other genome $D$ is the same, $d(E_2, D) = d(E_3, D)$. Only the matching to achieve this distance will be different, i.e. if the matching between $E_2$ and $D$ contains $(2_1, 2_i)$, the corresponding matching for $E_3$ and $D$ contains $(2_2, 2_i)$. We can exploit this redundancy to restrict the matching space as follows.

▶ **Lemma 8.** *Let $D_u$ and $D_v$ be degenerate genomes. Let marker $m$ occur $k$ times in $D_u$ and $n \geq k$ times in $D_v$, such that each occurrence in $D_v$ has the same adjacencies, i.e. $\{m_i^x, g_j^y\} \in D_v \implies \{m_{i'}^x, g_j^y\} \in D_v \forall 1 \leq i' \leq n$. Then for any linearisations $L_u, L_v$ with optimal matching $M$, there are linearisations $L'_u, L'_v$, such that their optimal matching $M'$ contains only pairs of the form $(m_i, m_i)$ and $d^s(L_u^M, L_v^M) = d^s(L_u'^{M'}, L_v'^{M'})$.*

**Proof.** By induction over $x(M) := |\{(m_i, m_j) \in M \mid i \neq j\}|$.

**Induction start:** Let $x(M) = 0$. Then $M$ is already a matching that conforms to the lemma, i.e. $L'_u = L_u$, $L'_v = L_v$ and $M' = M$.

**Induction step:** Let $x(M) > 0$. Let $(m_i, m_j) \in M$ with $i \neq j$. Then switch positions of $m_i$ and $m_j$ in $L_v$ to obtain $L''_v$. Then replace $(m_i, m_j)$ in $M$ by $(m_i, m_i)$ and if $(m_k, m_i) \in M$, replace it by $(m_k, m_j)$, obtaining matching $M''$. The distance is then the same, $d^s(L_u''^{M''}, L_v''^{M''}) = d^s(L_u^M, l_v^M)$, and $x(M'') \leq x(M) - 1$. ◀

For any internal node $u$ with degenerate genome $D_u$ and $n$ copies of marker $m$, we can thus find a neighbour $(u, v) \in E(T)$ that has $k \leq n$ copies of $m$ and allow only matchings of the type $(m_i, m_i)$ in the ILP solution presented in [7]. Note that we can do this only once per marker per internal node, that is restricting the matching space for more than one neighbour would lose optimality. For maximum effect, we chose the neighbour with the highest number of matching candidates.

## 3.4 Solving Strategy

While the weighting for Problem 2 modulates between an optimisation based on weights and an optimisation based on the distance and thus sacrifices optimality for minimising the overall distance in the tree, the problem tends to be easier to solve for $\alpha \to 0$. In fact, in [7], it is shown that finding a linearisation, can be reduced to a maximum weight matching problem, making it polynomial. We can use this situation to find solutions early.

Intuitively, adjacencies that are particularly frequent in a subtree – while not necessarily optimal – are probably not bad candidates as ancestral adjacencies either. We can use these frequencies as weights for Problem 2. Concretely, we use the weighted average of the number of occurrences for an adjacency at an internal node as its weight. We then first optimise versions of Problem 2 with $\alpha < 1$ before finally optimising with $\alpha = 1$. This ensures on hard instances of the problem that good solutions are found early.

A known heuristic strategy for SPP is to iteratively improve a solution by solving median problems at internal nodes [25]. We adapt this strategy to solving subtrees of the original tree first. In this step, the heuristic solution found in the weighted step is projected onto a subtree, used as an initial solution for the DCJ-SPP restricted to this subtree, and the DCJ-SPP for the subtree is solved. This gives us local solutions, and the lower bound as well as "variable hints" for the ILP solver from this subtree can be carried over to the problem on the full tree.

# 4 Evaluation

We implemented the Swofford-Maddison algorithm to determine marker numbers, the filtering of the solution space and the solving strategies as described in Section 3, integrating the implementation with the ILP developed in [7]. Importantly, we enforce the minimisation of $\delta|A^M \triangle B^M|$ not by including it in the objective function, but by requiring the cost for deleting markers is exactly the minimum determined previously by the Swofford-Maddison algorithm by a series of linear constraints.

We made the software available at `https://github.com/gi-bielefeld/spp_dcj_exact`. We call this software *SPP-DCJ-exact* in the following.

## 4.1 Simulated Data

We simulated data using Zombi [12]. Since our solution is intended to fit smaller phylogenies with a recent rearrangement history, we used scaled Zombi's default rearrangement parameters by a scale of 0.2, obtaining rates of 0.2 for duplications, 0.6 for gene losses, and 0.4 for inversions, transpositions and gene originations. By default, we simulated 10 lineages with the root genome containing 100 markers, unless specified otherwise. In the following, we will examine the impact of these parameters on SPP-DCJ-exact.

### 4.1.1 Performance

**Comparison with SPP-DCJ.** Recall that given the correct marker counts the naive solution to Small Parsimony for Natural Genomes (Problem 1) inputs the entire adjacency space into a linearisation problem optimizing for the total DCJ-indel distance. The latest solution for this problem is *SPP-DCJ-v2*, which consistently outperforms the original SPP-DCJ [7]. Since the space of adjacencies grows primarily with the number of markers, we varied the number of markers at the root genome of the phylogeny for this experiment. To that end, we simulated phylogenies with root genomes of 5 to 30 markers in increments of 5 with 10 samples per step and examined the solving time by gurobi 13.0 set to a time limit of 1 hour. In case no exact solution was found, we instead examined the *relative optimality gap*, that is the ratio $(b - l)/b$ between the current best solution $b$ and the tightest lower bound $l$ found by the solver. Recall that what we optimise for is the minimum number of rearrangements in the tree and thus $l \geq 0$ and a trivial solution is easily found. Therefore, we clamp the optimality gap to be between 0 and 100%. In this experiment, we did not use the solving strategies outlined in Section 3.4, neither for SPP-DCJ-v2 nor for SPP-DCJ-exact. The results are shown in Figure 3.

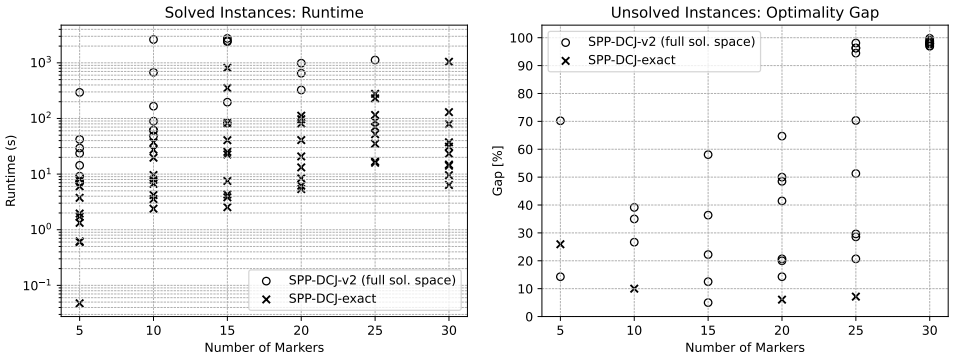



**Figure 3** Number of markers vs runtime or optimality gap of SPP-DCJ-v2 and SPP-DCJ-exact as solved by gurobi 13.0 with a time limit of 1 hour (= 3600s). Solved instances are shown on the left, giving their runtime. Instances that exceeded the time limit are shown on the right, instead giving the relative optimality gap reached within 1 hour.

We see that while both ILPs solving times rise with the number of markers, SPP-DCJ-exact outperforms SPP-DCJ-v2, finding exact solutions faster and if no exact solution is found, finding much closer approximations. Notably, almost all SPP-DCJ-v2 instances become unsolvable within the time limit once going above 20 markers, reaching over 90% relative optimality gaps for 30 markers, while the majority of instances of SPP-DCJ-exact remain solvable within 1 hour.

**Limiting Behaviour.** We examined the limits of which problem instances remain solvable using the strategies as detailed in Section 3.4. We evaluated three different settings: *Default*, in which we optimise using only gurobi with a set time limit; *Weights*, in which we allocate 30% of the time limit to solving the problem for $\alpha = 0.0, 0.1, 0.5$ before a final optimisation with the remaining time for $\alpha = 1$; *Weights+Subtrees*, in which we first perform the same weight based solves for 30% of the time limit and then switch to solve subtrees for 30% of the time limit, after which we run the final optimisation.

We performed three experiments, increasing the number of markers, lineages and rearrangements simulated by Zombi. These experiments are discussed in Appendix A.

To summarise the results, we find a majority of ILPs is solved within an hour up until about 600 markers, 15 lineages and up to 0.3 of the default rearrangement rate in Zombi. We also observe that using the "Weights" or "Weights+Subtree" strategies only starts to positively influence the solution for difficult problem instances and even has an adverse effect for simple instances. Moreover, the "Weights" strategy by itself is typically enough to obtain beneficial effects on optimality and through more time spent on the final optimisation may even outperform the "Weights+Subtrees" strategy.

One possible direction for future research is thus to examine whether these strategies can be improved by using more targeted approach, for example using a specific $\alpha$ or solving specific subtrees as well as allocating different ratios of solving time.

### 4.1.2 Reconstruction Quality

We simulated genomes using Zombi setting rearrangement rates to $0.2, 0.3, 0.4$ and $0.5$ of the default, generating 20 samples per step and ran SPP-DCJ-exact on a single thread with a time limit of 1 hour on each sample. For this experiment, we used the "Weights" strategy to guarantee a solution.

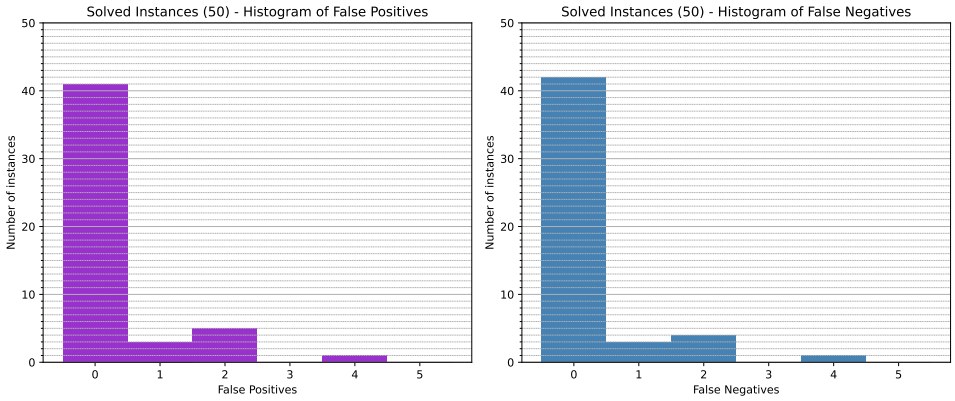



**Figure 4** Distribution of errors in reconstructed adjacencies. The histograms show the number of false positives (left) and false negatives (right) on the $x$-axis giving the number of solved instances with a given number of false positives or false negatives on the $y$-axis.

We then examined adjacencies at each internal node separately. Since duplicates complicate the comparison of adjacency sets, we limited ourselves in this analysis to adjacencies involving only non-duplicated markers. On average, each ground truth sample contained 780 such adjacencies. We then proceeded to evaluate *false positives* (FP), that is such adjacencies that are found in the reconstruction of SPP-DCJ-exact at a given node, but not in the ground truth simulated by Zombi and *false negatives* (FN), such adjacencies that are found in the ground truth, but not in the reconstruction.

The results for all solved instances are shown in Figure 4. We see that the reconstruction for the majority of samples is perfect – the adjacencies reconstructed by SPP-DCJ-exact correspond directly to adjacencies in the ground truth. In fact, all 41 samples without false positives also had no false negatives. Even when there are false reconstructions, the number of incorrectly reconstructed adjacencies observed in this experiment was at most 4 in a single instance.

For unsolved instances, we instead examined the relation between the relative optimality gap and the reconstruction quality. To that end, we calculated the *true positives* (TP), i.e. the adjacencies that are found in both the ground truth and the reconstruction at a given node. We then calculated *precision* as $TP/(TP + FP)$, *recall* as $TP/(TP + FN)$ and the *F1-score* as $2TP/(2TP + FP + FN)$. The results are visualised in Figure 5.

We see that, as expected, solution quality degenerates with a higher gap. However, overall, the quality of solutions remains high with precision and recall for all samples above 0.95, even when approaching a 100% gap. This suggests that SPP-DCJ-exact, at least when the input data has low to moderate numbers of rearrangements, may find optimal or near-optimal solutions, but that the lower bound does not always tighten fast enough to prove optimality.

## 4.2 Real Data

We applied our method to a set of 7 pathogenic plasmids from *Klebsiella pneumoniae*, previously characterised by Biggel *et al* in an epidemiological study of plasmids associated with bovine mastisis [1]. Sampling in this study was constrained to diseased Swiss cows over the course of two years, meaning the plasmid lineages within are recently related and relatively small. We selected the lineage "subcommunity 3" from this study, due to it being the most structurally diverse of the plasmid lineages.

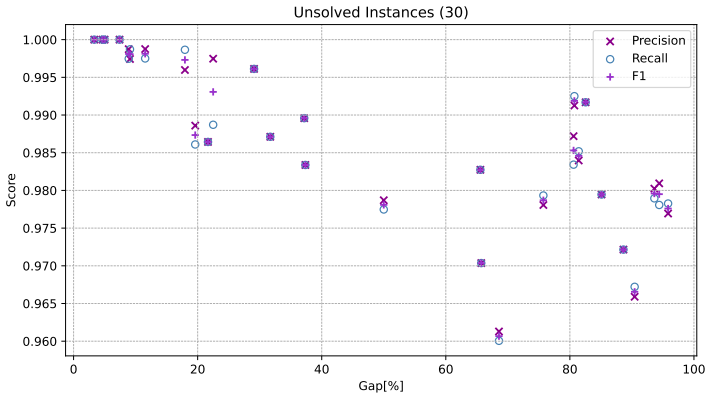



**Figure 5** Precision, recall and F1-score in adjacency reconstruction for unsolved instances by relative optimality gap after 1 hour of solving time.

We used MICE [6] to identify synteny blocks to use as markers, using a de Bruijn graph constructed using ggcat [10] with $k$-mer size 31 as input, and filtering out any synteny blocks below 200 bases. Statistics regarding the input genomes generated in this manner are found in Table 1 of Appendix B. We used the tree topology from Figure 2 in [1] as the input tree. We then applied SPP-DCJ-exact using both weighted and subtree strategies to determine ancestral sequences, using gurobi v11, 8 threads, and setting a time limit of a 12 hours. The best solution encompassed 28 DCJ-indel operations across the tree and was found after 70 minutes. Once the time limit was reached, a lower bound of 21 had been achieved, resulting in a relative optimality gap of 25%. The solution is visualised in Figure 6 with the help of gggenomes package in R [20].

All ancestral sequences are circular and unichromosomal, same as the extant plasmids. Their overall size is similar to that of the extant plasmids and, for cherry (n5(pK-40150-A,pK-75504-A)), the parental sequence n5 is larger than both its children. This demonstrates that in practice SPP-DCJ-exact does not "empty out" ancestral genomes, which is a direct result of restoring the triangle inequality. Additionally, there are multiple duplicate markers throughout the tree, including ones nested in rearrangement events spanning multiple markers, but nonetheless the solution parsimoniously matches these markers. This is particularly important for plasmids, as rearrangements are often associated with transposable elements flanked by duplicate genes [21]. From this solution, it is also possible to see that it is the same regions that are undergoing repeated rearrangements, e.g. the markers at the end of all the sequences are inverted multiple times across the whole tree. This is consistent with observations from previous studies of epidemiologically significant plasmids, which usually rely on manual reconstruction of rearrangement events [22, 31]. Overall, we can see in this example that our approach yields biologically plausible ancestral sequences, which can provide insights into the evolutionary history of a plasmid lineage.

## 5 Discussion

We introduced a unified framework for solving the Small Parsimony Problem for natural genomes that requires only extant genomes and the underlying tree as input, while still leveraging the efficient encoding of the solution space using degenerate genomes. In addition, we derived initial theoretical results that substantially restrict the solution space, thereby increasing the number of instances that can be solved exactly or approximated with strong guarantees.

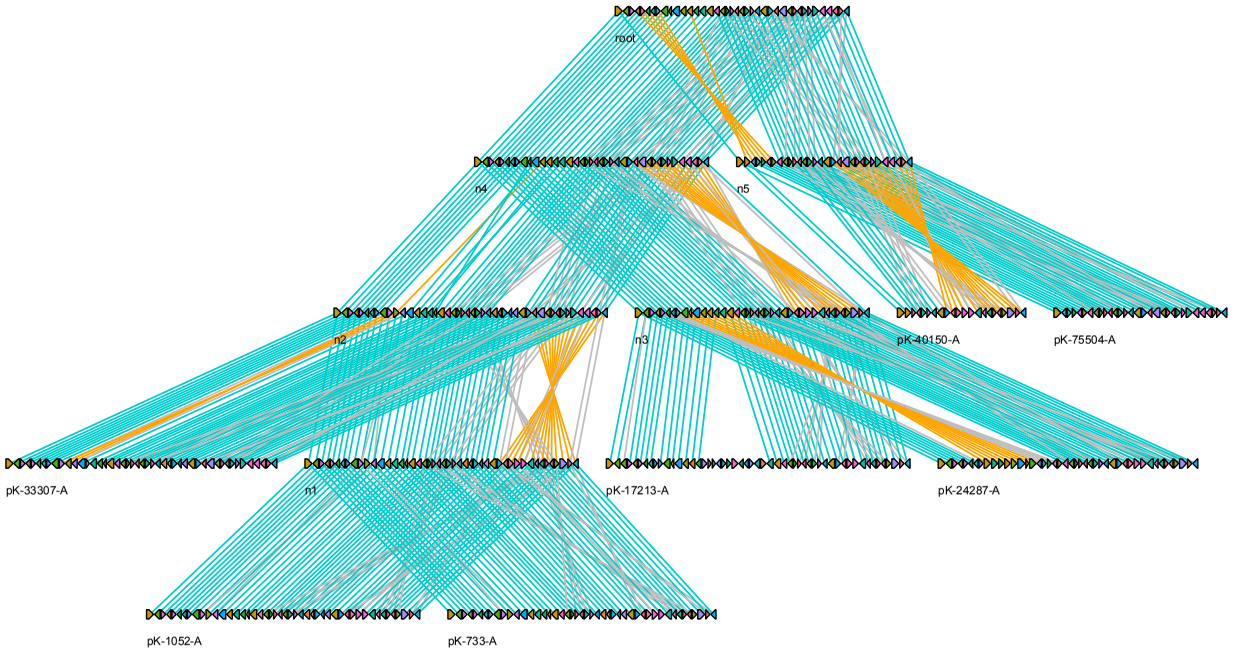



**Figure 6** Extant and reconstructed plasmid sequences, arranged according to their tree topology. The same markers across different sequences have the same colour, and marker orientation is shown by arrow direction. Teal links between markers mean they were matched and are oriented the same, orange links between markers mean they were matched and have opposite orientation, while grey lines show links between duplicate markers that were not matched in the final solution. Note that the selection of the root sequence is arbitrary; any intermediary genome between n4 and n5 could be the root, but here we simply choose n4.

When applied to both simulated data and real data sets with a recent evolutionary history, our approach consistently produces high-quality solutions within reasonable running times, even in cases where optimality was not conclusively proven.

The framework presented here also opens several promising directions for future research. So far, we have investigated only a limited number of simple, though effective, methods to constrain the solution space. A more detailed analysis of the underlying distance measure may allow for an even tighter pruning of degenerate genomes. Moreover, it would be interesting to study scenarios in which differences in marker content are not treated as the dominant term in the distance but can instead be traded off with rearrangement operations (i.e. "$\delta < \infty$"). Such an extension could naturally connect our framework to Duplication–Transfer–Loss models of gene tree parsimony, replacing the uniform insertion/deletion cost $\delta$ with operation-specific costs that distinguish between duplication, transfer, and loss events. Exploring these connections may further broaden the applicability of our approach.

## References


1 Michael Biggel, Magdalena Nüesch-Inderbinen, Mitsuko Logean, Sabrina Corti, Lucien Kelbert, and Roger Stephan. Genomic characterization of klebsiella spp. from bovine mastitis: dissemination of a conserved, highly transmissible lacacq+ fec+ plasmid drives burden of disease. *Applied and Environmental Microbiology*, 91(12):e01162–25, 2025. doi:10.1128/aem.01162-25.

2 Mathieu Blanchette, Takashi Kunisawa, and David Sankoff. Gene order breakpoint evidence in animal mitochondrial phylogeny. *Journal of Molecular Evolution*, 49(2):193–203, 1999. doi:10.1007/pl00006542.

3 Leonard Bohnenkämper. Recombinations, chains and caps: resolving problems with the DCJ-indel model. *Algorithms for Molecular Biology*, 19:8, 2024. doi:10.1186/s13015-024-00253-7.

4 Leonard Bohnenkämper and Daria Frolova. gi-bielefeld/spp_dcj_exact. Software, swhId: swh:1:dir:9d2d2213b28d627478afc8a5f365b31b0d0979f1 (visited on 2026-08-15). URL: https://github.com/gi-bielefeld/spp_dcj_exact, doi:10.4230/artifacts.27684.

5 Leonard Bohnenkämper, Marília D.V. Braga, Daniel Doerr, and Jens Stoye. Computing the rearrangement distance of natural genomes. *Journal of Computational Biology*, 28(4):410–431, 2021. doi:10.1089/cmb.2020.0434.

6 Leonard Bohnenkämper, Luca Parmigiani, Cedric Chauve, and Jens Stoye. On deriving synteny blocks by compacting elements. *bioRxiv*, page 2025.12.15.694404, 2026. doi:10.64898/2025.12.15.694404.

7 Leonard Bohnenkämper, Jens Stoye, and Daniel Doerr. Reconstructing rearrangement phylogenies of natural genomes. *Algorithms for Molecular Biology*, 20(1):10, 2025. doi:10.1186/s13015-025-00279-5.

8 Marília D.V. Braga, Eyla Willing, and Jens Stoye. Double cut and join with insertions and deletions. *Journal of Computational Biology*, 18(9):1167–1184, 2011. doi:10.1089/cmb.2011.0118.

9 Alberto Caprara. Formulations and hardness of multiple sorting by reversals. In *Proceedings of the third annual international conference on Computational molecular biology*, pages 84–93, 1999. doi:10.1145/299432.299461.

10 Andrea Cracco and Alexandru I Tomescu. Extremely fast construction and querying of compacted and colored de bruijn graphs with ggcat. *Genome Research*, 33(7):1198–1207, 2023.

11 Elise Darmon and David R. F. Leach. Bacterial genome instability. *Microbiology and Molecular Biology Reviews*, 78(1):1–39, 2014. doi:10.1128/mmbr.00035-13.

12 Adrián A Davín, Théo Tricou, Eric Tannier, Damien M de Vienne, and Gergely J Szöllősi. Zombi: a phylogenetic simulator of trees, genomes and sequences that accounts for dead linages. *Bioinformatics*, 36(4):1286–1288, 2019. doi:10.1093/bioinformatics/btz710.

13 Daniel Doerr and Cedric Chauve. Small parsimony for natural genomes in the DCJ-indel model. *Journal of Bioinformatics and Computational Biology*, 19(06):2140009, 2021. doi:10.1142/s0219720021400096.

14 Yucheng Dong, Qi He, Xinyu Chen, Fan Yang, Li He, and Yongchang Zheng. Extrachromosomal DNA (ecDNA) in cancer: mechanisms, functions, and clinical implications. *Frontiers in Oncology*, 13:1194405, 2023. doi:10.3389/fonc.2023.1194405.

15 Wandrille Duchemin, Yoann Anselmetti, Murray Patterson, Yann Ponty, Sèverine Bérard, Cedric Chauve, Celine Scornavacca, Vincent Daubin, and Eric Tannier. DeCoSTAR: Reconstructing the ancestral organization of genes or genomes using reconciled phylogenies. *Genome Biology and Evolution*, 9(5):1312–1319, 2017. doi:10.1093/gbe/evx069.

16 James S Farris. Methods for computing Wagner trees. *Systematic zoology*, pages 83–92, 1970.

17 Pedro Feijao and Joao Meidanis. Scj: A breakpoint-like distance that simplifies several rearrangement problems. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 8(5):1318–1329, 2011. doi:10.1109/TCBB.2011.34.

18 Walter M. Fitch. Toward defining the course of evolution: Minimum change for a specific tree topology. *Systematic Biology*, 20(4):406–416, 1971. doi:10.1093/sysbio/20.4.406.

19 Daria Frolova. *Studying plasmid relatedness and evolution through structural variation*. PhD thesis, Apollo - University of Cambridge Repository, 2025. doi:10.17863/CAM.123913.

20 Thomas Hackl, Markus Ankenbrand, Bart van Adrichem, David Wilkins, and Kristina Haslinger. gggenomes: effective and versatile visualizations for comparative genomics. *arXiv*, 2024. doi:10.48550/arxiv.2411.13556.

21 Christopher J. Harmer, Robert A. Moran, and Ruth M. Hall. Movement of IS26-associated antibiotic resistance genes occurs via a translocatable unit that includes a single IS26 and preferentially inserts adjacent to another IS26. *mBio*, 5(5):10.1128/mbio.01801–14, 2014. doi:10.1128/mbio.01801-14.

22 Timothy J. Johnson, Jessica L. Danzeisen, Bonnie Youmans, Kyle Case, Katharine Llop, Jeannette Munoz-Aguayo, Cristian Flores-Figueroa, Maliha Aziz, Nicole Stoesser, Evgeni Sokurenko, Lance B. Price, and James R. Johnson. Separate f-type plasmids have shaped the evolution of the h30 subclone of escherichia coli sequence type 131. *mSphere*, 1(4):10.1128/msphere.00121–16, 2016. doi:10.1128/msphere.00121-16.

23 Ankit Malhotra, Michael Lindberg, Gregory G. Faust, Mitchell L. Leibowitz, Royden A. Clark, Ryan M. Layer, Aaron R. Quinlan, and Ira M. Hall. Breakpoint profiling of 64 cancer genomes reveals numerous complex rearrangements spawned by homology-independent mechanisms. *Genome Research*, 23(5):762–776, 2013. doi:10.1101/gr.143677.112.

24 William Matlock, Liam P Shaw, Samuel K Sheppard, and Edward Feil. Towards quantifying plasmid similarity. *Microbial Genomics*, 10(9), 2024. doi:10.1099/mgen.0.001290.

25 David Sankoff and Mathieu Blanchette. Multiple genome rearrangement and breakpoint phylogeny. *Journal of computational biology*, 5(3):555–570, 1998. doi:10.1089/CMB.1998.5.555.

26 Mingfu Shao, Yu Lin, and Bernard M.E. Moret. An exact algorithm to compute the double-cut-and-join distance for genomes with duplicate genes. *Journal of Computational Biology*, 22(5):425–435, 2015. PMID: 25517208. doi:10.1089/cmb.2014.0096.

27 Eva H. Stukenbrock and Daniel Croll. The evolving fungal genome. *Fungal Biology Reviews*, 28(1):1–12, 2014. doi:10.1016/j.fbr.2014.02.001.

28 David L. Swofford and Wayne P. Maddison. Reconstructing ancestral character states under Wagner parsimony. *Mathematical Biosciences*, 87(2):199–229, 1987. doi:10.1016/0025-5564(87)90074-5.

29 Eric Tannier, Chunfang Zheng, and David Sankoff. Multichromosomal median and halving problems under different genomic distances. *BMC Bioinformatics*, 10(1):120, April 2009. doi:10.1186/1471-2105-10-120.

30 Gerry Tonkin-Hill, Jukka Corander, and Julian Parkhill. Challenges in prokaryote pangenomics. *Microbial Genomics*, 9(5):mgen001021, 2023. doi:10.1099/mgen.0.001021.

31 Ethan R. Wyrsch, Rhys N. Bushell, Marc S. Marenda, Glenn F. Browning, and Steven P. Djordjevic. Global phylogeny and f virulence plasmid carriage in pandemic escherichia coli ST1193. *Microbiology Spectrum*, 10(6):e02554–22, 2022. doi:10.1128/spectrum.02554-22.

32 Wei Xu, Daniel Jameson, Bin Tang, and Paul G. Higgs. The relationship between the rate of molecular evolution and the rate of genome rearrangement in animal mitochondrial genomes. *Journal of Molecular Evolution*, 63(3):375–392, 2006. doi:10.1007/s00239-005-0246-5.

# A Evaluating Limiting Behaviour of Solving Strategies

To assess the performance of the three strategies introduced in Section 4.1.1 (*Default*, *Weights*, and *Weights+Subtrees*), we evaluated their relative optimality gap under a time limit of one hour. We considered three parameters: (i) the number of markers in the root genome, (ii) the number of lineages, and (iii) the number of rearrangements. For each parameter configuration, we generated 10 samples using Zombi. We solved each problem instance with all three strategies, each running on a single thread.

In the first experiment, we increased the number of markers in the root genome from 200 to 1000 in increments of 100. As shown in Figure 7 (A), the optimality gap increases with the number of markers for all three strategies. For small genomes (200–500 markers), most instances are solved optimally within the time limit. Somewhat unexpectedly, the *Default* strategy performed best in this setting, outperforming both *Weights* and *Weights+Subtrees*. A possible explanation is that, for larger genomes, the weighted and subtree-based approaches generate subproblems that themselves become computationally expensive. The additional overhead needed to solve these subproblems may thus "absorb" solving time that would be better used solving the original problem.

In the second experiment, we increased the number of lineages from 11 to 20 in increments of 1. The results are visualised in Figure 7 (B). Across all strategies, the variance in the observed optimality gaps is high. Up to 15 lineages, the majority of instances are solved optimally within the time limit. For 19 and 20 lineages onwards, the *Weights* strategy begins to show an improvement over *Default*. The *Weights+Subtrees* strategy surpasses *Default* only for some instances with a high number of lineages. Notably, however, it exhibits a considerably smaller variance in optimality gaps than both other strategies at these higher values, indicating more stable performance on particularly difficult instances.

In the third experiment, we varied the number of rearrangements. To that end, we scaled duplication rates by $x$, gene loss rates by $3x$, and other rearrangement rates by $2s$, increasing $s$ from 0.2 to 1.0 in increments of 0.1. We call $s$ *rearrangement scale* in the following. As shown in Figure 7 (C), the optimality gap increased sharply with growing rearrangement rates. Most problem instances are solved optimally for $s \leq 0.4$. From $s = 0.7$ onwards, the *Default* strategy almost exclusively exhibits gaps of 100%, indicating that either no solution was found and/or the lower bound could not be tightened further. In the intermediate range $s = 0.6$ to 0.8, *Weights+Subtrees* outperforms *Weights*, but only on some instances. For higher rates of rearrangements, however, the relative optimality gap of all strategies approaches 100%, suggesting the overall time limit of 1 hour is not enough to obtain any significant information about a given problem instance.

Overall, these results indicate that *Weights* and *Weights+Subtrees* are primarily beneficial strategies for solving difficult problem instances. For simple problem instances, the additional preprocessing and decomposition effort appears unnecessary and may even reduce performance. This suggests that further fine-tuning is required, in particular mechanisms for identifying in advance whether solving particular subtrees or applying a weighted optimisation first is likely to improve performance.

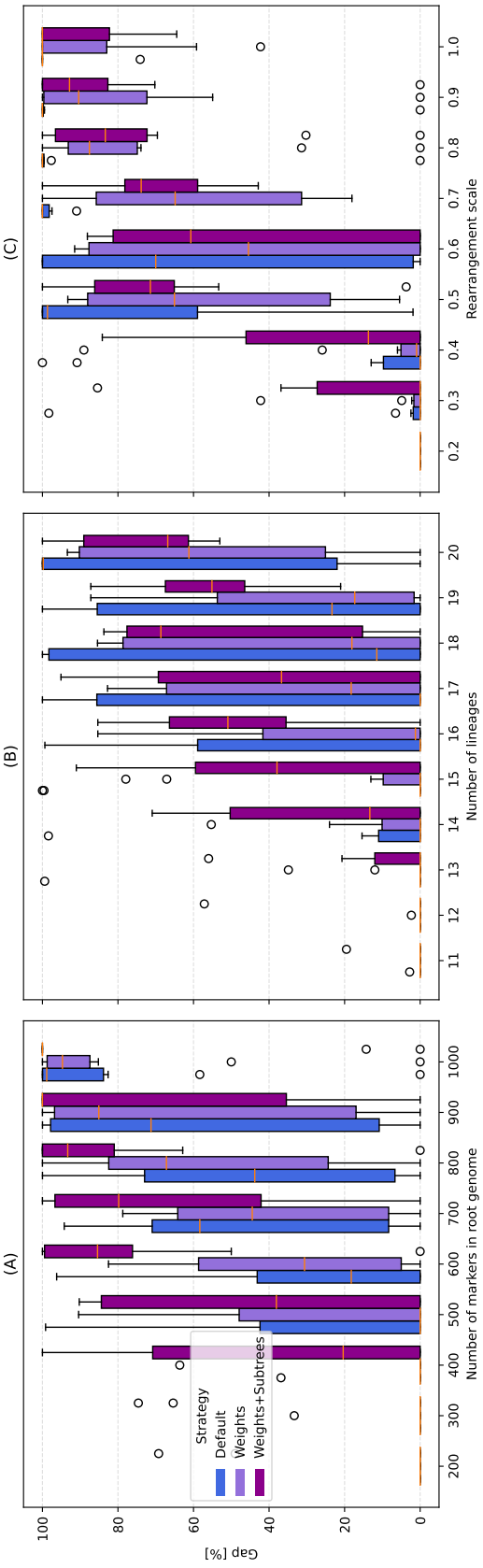



**Figure 7** Distribution of optimality gaps for three solving strategies (Default, Weights and Weights+Subtrees) after 1 hour of solving time using gurobi 13.0 in three experiments simulated with Zombi. Relative optimality gaps are visualised as a box plot. (A) The number of markers in the root genome is increased from 200 to 1000 in steps of 100. (B) The number of simulated lineages is increased from 11 to 20. (C) The default rearrangement rate is scaled from 0.2 to 1.0 in increments of 0.1. For each step in each of the experiments 10 samples were generated.

## B Plasmid Marker Statistics

**Table 1** Summarising statistics of the MICE output used for marker definition. The first column is the plasmid sequence ID, followed by the number of marker occurrences, followed by number of singular markers, and finally the number of duplicate markers.

| seq_id | length | # singular markers | # duplicate markers |
|---|---|---|---|
| pK-75504-A | 31 | 15 | 8 |
| pK-1052-A | 47 | 29 | 9 |
| pK-33307-A | 47 | 29 | 9 |
| pK-40150-A | 22 | 8 | 7 |
| pK-24287-A | 45 | 19 | 13 |
| pK-733-A | 46 | 28 | 9 |
| pK-17213-A | 53 | 31 | 11 |

# DivQuant: Estimation of Species Richness and Entropy from Small Samples

**Johanna Elena Schmitz**
Algorithmic Bioinformatics, Faculty of Mathematics and Computer Science,
Saarland University, Saarbrücken, Germany
Saarbrücken Graduate School of Computer Science, Germany
Center for Bioinformatics Saar, Saarland Informatics Campus, Saarbrücken, Germany

**Sven Rahmann**[1]
Algorithmic Bioinformatics, Faculty of Mathematics and Computer Science,
Saarland University, Saarbrücken, Germany
Center for Bioinformatics Saar, Saarland Informatics Campus, Saarbrücken, Germany

## Abstract

Estimating diversity properties of discrete distributions from a small observed sample is a fundamental problem in algorithmic statistics that has applications in many fields, in particular bioinformatics, but also in ecology or linguistics. The two most common diversity measures are the number of distinct elements in a multiset, also referred to as "species richness" in ecology or "alpha diversity" in microbial analysis, and the Shannon entropy, also referred to as "evenness". Estimating these properties from a small sample is particularly challenging for distributions with many rare elements. Thus, many estimators have been proposed in the past that, in practice, work well for different types of distributions.

We present DivQuant, an optimization-based, extrapolating richness and entropy estimator with three contributions. First, we formulate the upsampling problem as a convex quadratic program with a Neyman $\chi^2$ objective. Unlike the linear program of its predecessor RichnEst, DivQuant admits confidence intervals via $\chi^2$ test inversion that are empirically well-calibrated. Second, we replace RichnEst's fixed-threshold fingerprint truncation with the rare/abundant fingerprint split of Valiant and Valiant, which strongly reduces problem size and preserves enough degrees of freedom for the confidence-interval program to remain valid and feasible. Third, we plug the optimal population fingerprint returned by the program into Shannon's entropy formula to obtain an entropy estimate.

DivQuant attains close-to-nominal 95% confidence intervals in essentially all tested regimes, including six simulated distribution families, *Tara Oceans* microbiome data, and *10X Genomics* scRNA-seq data, while competing state-of-the-art methods (RichnEst, iNext, PreSeq) miss the true richness in up to 80% of instances, well above the nominal 5%. In addition, DivQuant outperforms classical asymptotic entropy estimators (Miller-Madow, CAE) and the extrapolating iNext estimator. Running times remain competitive, with DivQuant typically completing in seconds.

**2012 ACM Subject Classification** Mathematics of computing → Probability and statistics; Mathematics of computing → Linear programming; Mathematics of computing → Quadratic programming; Applied computing → Bioinformatics

**Keywords and phrases** diversity estimation, alpha diversity, species richness, entropy estimation, upsampling, linear program, quadratic program

**Digital Object Identifier** 10.4230/LIPIcs.WABI.2026.16

**Supplementary Material** *Software (Source Code)*: `https://gitlab.com/rahmannlab/divquant` [37] archived at `swh:1:dir:1ca96b0e0bbf05c5e7afa9dee8101fd46e1246b5`

[1] Corresponding author



© Johanna Elena Schmitz and Sven Rahmann;
licensed under Creative Commons License CC-BY 4.0



## 1 Introduction

With access to only a small sample from a large population, there are many questions one may ask about the total population. For example, how many distinct elements went unseen in the sample? Or, how even is the population, i.e., what is the composition of the population regarding rare and abundant elements? The term *richness* is often used to denote the number of distinct elements in a population or sample. Evenness is measured using *entropy*, often referred to as Shannon diversity [39]. Both measures are important diversity measures in many fields, such as ecology [16, 42], linguistics [19, 20] and biology [5]. In microbial sequence analysis, richness (also called alpha diversity) and entropy are commonly used to analyse bacterial communities or environments [44, 41], where both function and community dynamics are assessed based on diversity changes over time. Furthermore, both measures are used to evaluate the diversity of immune repertoires or to track changes in the immune system during health and disease [18, 34].

However, there is an open debate about how best to compare the diversity between samples and which diversity measure to use [6]. Richness is usually computed using either the observed sample richness, which underestimates the true richness for almost all samples, or by using a statistical estimate. The same is true for entropy. An overview of the sampling problem is shown in Figure 1A–C.

**Related work.** Richness and entropy estimators can be divided into two groups depending on whether they require the full population size to be known or not, referred to as extrapolating and asymptotic estimators, respectively [9]. Ecological studies typically require accurate asymptotic lower bounds or estimates for extrapolated populations that are up to three times larger than the observed sample. Other applications, such as metagenomics or immune repertoire diversity, require accurate estimates for populations that are more than ten times larger than the observed sample.

Some asymptotic estimators, like the Chao 1 [8] or Jackknife estimator [7], assume that the number of species seen only once or twice is most predictive of the number of unseen species. Other estimators assume that the sample was drawn from a parametric family of probability distributions, e.g., a Gamma-Poisson mixture distribution, and reduce the problem to estimating the parameters of a parametric probability distribution, using method of moments, maximum likelihood, or Bayesian estimators [16, 27, 11, 3]. Several recent methods make no such assumptions and are based on linear programming [40, 38], link Hill numbers to rarefaction/extrapolation sampling curves (richness corresponds to Hill numbers of order 0) [24], perform curve fitting on the rarefaction curve [14, 28] or on ratios of consecutive frequency counts [43]. A recent comparative review by Schmitz and Rahmann [36] shows that despite this long history, there remains an unmet demand for accurate richness estimators when the population is more than ten times larger than the observed sample.

For entropy, classical asymptotic estimators include the Miller-Madow estimator [30] and the coverage-adjusted estimator (CAE) [10], complemented by more recent linear-programming-based estimators [40]. Because Hill numbers of order 1 correspond to the exponential of the Shannon diversity, iNEXT [24] can also be used to extrapolate entropy. For a broader overview of entropy estimators, we refer the reader to the survey by Paninski [33].

Two practical issues complicate richness (and entropy) estimation in DNA sequencing applications. First, sample diversity (in particular, sample richness) is highly correlated with sequencing depth, which has led to the widespread practice of comparing samples after rarefaction [15, 35]. However, rarefaction discards information, so extrapolating estimators

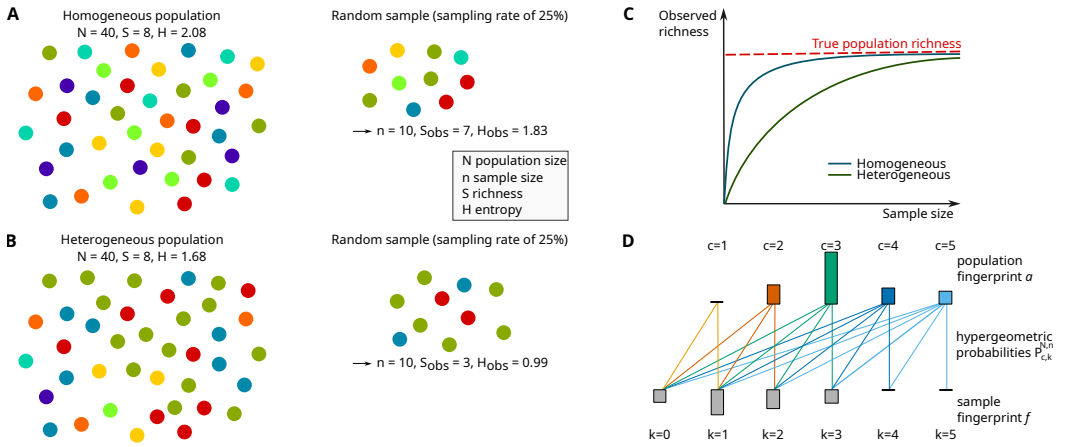



**Figure 1** Difference between population richness $S$ (entropy $H$) and observed sample richness (entropy). Colours denote distinct species. **A** In a homogeneous population, a small sample suffices to observe a richness close to the true value. **B** For a population with many rare species, the observed sample richness underestimates the population richness. **C** Species richness is commonly assessed via a rarefaction curve: as the sample size grows, the observed richness converges to the population richness, at a rate that depends on the species composition. **D** Example of the downsampling process from a population fingerprint $a$ to a sample fingerprint $f$. A species observed 3 times in the sample must have abundance $\geq 3$ in the population, so the expected value of $f_3$ is $a_3 \cdot P_{3,3}^{N,n} + a_4 \cdot P_{4,3}^{N,n} + a_5 \cdot P_{5,3}^{N,n}$ (see Section 2.2). Figure adapted from [36].

are preferable in many settings [13, 9, 25]. Second, sequencing errors inflate the singleton count, leading to biased estimates. Many standard error correction methods remove all singletons, which violates the assumptions of estimators such as Chao 1 [12, 6]. These two issues increase the demand for estimators that are less dependent on the sequencing depth and the number of singletons.

**The uncertainty quantification gap.** A diversity estimate is only as useful as the uncertainty attached to it, since many downstream questions are comparison tasks for which a point estimate alone does not yield valid conclusions. Yet uncertainty quantification has received much less attention than point-estimate accuracy in the above-mentioned literature. Several widely used entropy estimators (Miller-Madow, CAE, Unseen) report no confidence intervals at all. Among extrapolating estimators that do, iNEXT provides analytic intervals, PreSeq provides bootstrap intervals, and RichnEst provides intervals derived from a per-term linear proxy, but to our knowledge none has been systematically shown to achieve nominal coverage in the regime where the population is more than ten times larger than the observed sample. We show in Section 3 that this gap is sharp: across six simulated distribution families and two biological datasets, the nominal 95% intervals of iNEXT, PreSeq, and RichnEst miss the truth in up to 80% of instances, well above the nominal 5%.

**Contributions.** We close this gap with DivQuant, an extrapolating richness and entropy estimator whose 95% confidence intervals empirically cover the truth in essentially all tested regimes. The core change is methodological: we replace the linear objective of DivQuant's predecessor RichnEst [38], which searches via a linear program for the most plausible population histogram consistent with the observed sample under hypergeometric subsampling, with a quadratic Neyman $\chi^2$ objective [32], turning the search into a convex

quadratic program whose statistic is asymptotically $\chi^2$ distributed under certain assumptions. Minimizing and maximizing the population richness over the set of fingerprints whose $\chi^2$ statistic falls below the desired quantile yields confidence intervals. We contribute two supplementary enhancements.

- We replace RichnEst's fixed-threshold sample truncation with the rare/abundant fingerprint split of Valiant and Valiant [40]. The fixed threshold collapses the degrees of freedom of the $\chi^2$ statistic and frequently renders the confidence-interval program infeasible; the rare/abundant split preserves enough degrees of freedom for the program to remain feasible across a much wider range of inputs.
- We plug the optimal population fingerprint returned by the program into Shannon's entropy formula to obtain a point estimate, and reuse the same min/max-over-feasible-set construction to obtain entropy confidence intervals. The resulting point estimates outperform classical asymptotic estimators (Miller-Madow, CAE, Unseen) and the extrapolating iNEXT estimator on both simulated and real data, and the entropy intervals cover the truth in a larger proportion of cases than iNEXT's.

In the next section, we describe DivQuant in more detail. In Section 3, we compare DivQuant with state-of-the-art estimators for both richness estimation and entropy estimation on simulated and biological data. We give concluding remarks in Section 4.

# 2 Methods

After introducing the notation, we describe our assumptions and explain how to formulate the estimation problem as a convex optimization problem.

## 2.1 Notation

We are interested in a population with $N$ elements (sometimes called individuals) but have only access to a small random sample $x = (x_1, x_2, \dots, x_n)$ of size $n < N$. Each element belongs to a single species (sometimes called class or group), and the richness $S$ of a population is given by the number of distinct elements (or species).

Assuming that the species have relative frequencies or probabilities $p = (p_1, p_2, \dots, p_S)$ in the population, we can compute the (Shannon) entropy by

$$H_p := -\sum_{i=1}^{S} p_i \log p_i$$

where, depending on the research area, it is common to use $\log_2$, $\log_{10}$ or $\log_e$.

Both species richness and entropy are invariant under relabeling; we will thus in the following consider the *fingerprint* of a sample (sometimes called *frequency* or *occupancy* vector). Given a sample $x = (x_1, x_2, \dots, x_n)$, the $k$-th component of the fingerprint $f = (f_1, f_2, \dots)$ counts how many species occurred exactly $k$ times in the sample. Note that $f_0$ corresponds to the unseen species. Thus, the sample size satisfies

$$n = \sum_{k \geq 1} k\, f_k\,,$$

and the observed richness is given by

$$S_{\text{obs}} = \sum_{k \geq 1} f_k\,.$$

**Table 1** Notation.

| Symbol | Meaning |
|---|---|
| $N$, $n$ | population size and sample size ($n < N$) |
| $S$, $S_{\text{obs}}$ | population richness and observed sample richness |
| $H$ | Shannon entropy |
| $x = (x_1, \ldots, x_n)$ | sample (one entry per element) |
| $f = (f_1, \ldots, f_{K_{\text{obs}}})$ | sample fingerprint: $f_k$ is the number of species observed exactly $k$ times |
| $f_0$ | number of unseen species |
| $K_{\text{obs}}$ | largest count actually observed in $f$ |
| $a = (a_1, \ldots, a_C)$ | population fingerprint: $a_c$ is the number of species with abundance $c$ |
| $f' = (f'_0, f'_1, \ldots, f'_K)$ | expected sample fingerprint under hypergeometric subsampling from $a$ |
| $C$, $K$ | truncation parameters for $a$ and $f'$ (Section 2.5) |
| $P_{c,k}^{N,n}$ | hypergeometric probability that a species with abundance $c$ is observed exactly $k$ times when downsampling from $N$ to $n$ elements (Eq. 2) |

Given the fingerprint, the (empirical) sample entropy can be expressed as

$$H_f = -\sum_{k \geq 1} f_k \frac{k}{n} \log \frac{k}{n}, \tag{1}$$

where each species' probability is approximated by its observed frequency $k/n$. The symbols used throughout the paper are collected in Table 1.

## 2.2 Sampling Process

We adopt the same probabilistic model as RichnEst [38].

The observed fingerprint $f$ is assumed to arise from drawing a random sample of size $n$ from the population. Thus, our goal is to find a population of size $N$ (or in practice, a larger sample of interest with size $N$), whose *expected* fingerprint $f'$ is close to the *observed* fingerprint $f$.

A species that occurs $c$ times in the population can be observed $0 \leq k \leq c$ times in the sample. The probability that a species with $c$ individuals in a population of size $N$ is observed exactly $k$ times in a sample of size $n$ is given by the hypergeometric probability

$$P_{c,k}^{N,n} = \frac{\binom{c}{k}\binom{N-c}{n-k}}{\binom{N}{n}}. \tag{2}$$

Given a population fingerprint $a = (a_1, a_2, \ldots, a_C)$, where the most frequent species has $C$ individuals and $\sum_{c=1}^{C} c\, a_c = N$, the expected fingerprint $f' = (f'_0, f'_1, f'_2, \ldots, f'_K)$ of a sample of size $n$ is

$$f'_k = \sum_{c=k}^{C} a_c\, P_{c,k}^{N,n} \quad (0 \leq k \leq K \leq C).$$

See Figure 1D for an example. We exploit below that the contribution to $f'_k$ from a species with abundance $c$, namely $a_c\, P_{c,k}^{N,n}$, is the mean of a Binomial$(a_c, P_{c,k}^{N,n})$ random variable. When the per-trial success probabilities $P_{c,k}^{N,n}$ are small, the sum of Binomial random variables is well approximated by a Poisson distribution due to Le Cam's theorem [29, 23].

## 2.3 Estimation Problem

We estimate the species richness by modelling the inverse sampling process to find the most plausible population fingerprint $a$ that can explain the observed sample fingerprint $f = (f_1, f_2, \dots)$. The richness estimate is then $\hat{S} = \sum_{c=1}^{C} a_c$.

We formulate the problem as a convex optimization problem and turn our goal that the expected fingerprint $f'$ should be close to the observed fingerprint $f$ into the objective function. We adopt the minimum $\chi^2$ method [32, 4], which is widely used for parameter estimation in counting experiments whose histogram entries are approximately Poisson distributed [22, 26] and for comparing histograms [2, 17]. Concretely, we use Neyman's $\chi^2$ statistic [32]

$$\chi^2_{\text{Neyman}} = \sum_{k=1}^{K} (f_k - f'_k)^2 / \max(f_k, 1) \tag{3}$$

Following common practice, we use $\max(f_k, 1)$ in the denominator to handle zero counts [22]. Equation (3) is asymptotically $\chi^2$ distributed if it is approximately a sum of squared independent standard normal variables. If each bin is Poisson distributed, the observed count $f_k$ in the denominator consistently estimates the variance of $f_k$. For sufficiently large bin counts, the standardized deviation $(f_k - f'_k)/\sqrt{\max(f_k, 1)}$ is then approximately standard normal. This allows us to use the $\chi^2$ statistic to compute confidence intervals in Section 2.4 [2, 26]. Compared to Pearson's $\chi^2$ (which uses $f'_k$ in the denominator), Neyman's variant has the advantage of yielding a quadratic objective in the variables, hence preserving convexity of the program below. In a good solution, $f_k \approx f'_k$, and the difference is marginal.

Given the observed sample fingerprint $f = (f_1, f_2, \dots)$ of size $n$ and the population size $N$, we solve the following convex optimization problem to search for the population fingerprint $a = (a_1, a_2, \dots, a_C)$, where $P^{N,n}_{c,k}$ are the hypergeometric probabilities (constant during solving) of Equation (2) and the vector $f' = (f'_0, f'_1, \dots, f'_K)$ are auxiliary variables that store the expected fingerprint. All variables are constrained to be non-negative.

$$\begin{aligned}
&\text{minimize} && \sum_{k=1}^{K} (f_k - f'_k)^2 / \max(f_k, 1) && \\
&\text{subject to} && && \\
&\quad f'_k = && \sum_{c=k}^{C} a_c \cdot P^{N,n}_{c,k} && (k = 0, \dots, K) \\
&\quad f'_0 = && \sum_{c=1}^{C} a_c - \sum_{k=1}^{K} f_k && \\
&\quad N = && \sum_{c=1}^{C} c \cdot a_c && \\
&\quad a_c \geq && 0 && (c = 1, \dots, C)
\end{aligned}$$

The important variables are the components of the population fingerprint $a$, which determine $f'$, which has to be close to the observed $f$. The optimization problem formulation depends on the truncation parameters $C$ and $K$, which we do not know initially. We describe their choice in Section 2.5. Instead of penalizing the deviation between the *expected* and the *observed* number of unseen elements in the objective function, we add both as hard constraints to the optimization problem (i.e., $f'_0 = \sum_{c=0}^{C} a_c \cdot P^{N,n}_{c,0}$ and $f'_0 = \hat{S} - S_{\text{obs}}$). We

relax both $a$ and the auxiliary variables $f'$ to be non-negative real numbers rather than integers. While this does not yield strictly valid fingerprints, it makes the program more efficient while empirically leaving the richness estimate essentially unchanged. There may exist several solutions for $a$ with (almost) equivalent objective values; we come back to this below. The formulation of RichnEst uses the linear proxy $\sum_{k=1}^{K} |f_k - f'_k|/\sqrt{f_k + 1}$, which in practice gives similar point estimates but makes it hard to derive confidence intervals [38].

## 2.4 Confidence Intervals

Since many population fingerprints $a$ can yield comparable objective values, a confidence interval for the richness estimate is more informative than a single point estimate. We construct the confidence interval by inverting the $\chi^2$ test.

### 2.4.1 Inverted $\chi^2$ Test

The general idea of inverting a $\chi^2$ test is to collect all parameter values under which the hypothesis would not be rejected [1].

For each candidate $a$, the null hypothesis is that $a$ is the true population fingerprint. Thus, inverting the tests gives a confidence region for the true $a$. Let $\mathcal{A}$ be the set of valid population fingerprints whose Neyman $\chi^2$ statistic, computed between their expected fingerprint $f'$ and the observed fingerprint $f$, is below the critical value $\varepsilon$ at the desired confidence level. The minimum and maximum of $\sum_{c=1}^{C} a_c$ over $a \in \mathcal{A}$ then form a confidence interval for the true richness $S$.

We therefore minimize or maximize the richness under the constraint that the deviation between the observed and expected fingerprint is bounded by $\varepsilon$, whose choice is discussed below. For the lower bound, we thus solve the quadratically constrained linear program

$$\text{minimize} \quad \sum_{c=1}^{C} a_c \tag{4}$$

subject to

$$\varepsilon \geq \sum_{k=1}^{K} (f_k - f'_k)^2 / \max(f_k, 1) \tag{5}$$

$$f'_k = \sum_{c=k}^{C} a_c \cdot P^{N,n}_{c,k} \qquad (k = 0, \ldots, K) \tag{6}$$

$$f'_0 = \sum_{c=1}^{C} a_c - \sum_{k=1}^{K} f_k \tag{7}$$

$$N = \sum_{c=1}^{C} c \cdot a_c \tag{8}$$

$$a_c \geq 0 \qquad (c = 1, \ldots, C) \tag{9}$$

and, respectively, maximize objective (4) to get the upper bound. We use the same non-negativity constraints and relaxation to real numbers for the variables as described in Section 2.3. The optimization problem is convex because all equality constraints are affine and the inequality constraint (5) is convex.

We set $\varepsilon$ to the value of the quantile function of a $\chi^2$ distribution with $K - 1$ degrees of freedom for the desired confidence level (default 0.95). This is a heuristic choice, but one for which there are good reasons. Under the null hypothesis, by Le Cam's theorem,

the $(f_k)_{1\le k\le K}$ are approximately independent Poisson variables [29] linked by the single constraint $\sum_{k=1}^{K} k\, f_k = n$. Thus, Neyman's $\chi^2$ statistic asymptotically follows a $\chi^2_{K-1}$ distribution, see Section 2.3. The unseen species $f'_0$ do not enter the $\chi^2$ sum and hence do not affect the degrees of freedom. Although the optimization later searches over $a$, the test inversion fixes $a$ when evaluating the statistic: for each candidate $a$ we ask whether the observed $f$ is compatible with $a$ as a fixed null. Because $a$ is not estimated from $f$, the usual degrees-of-freedom penalty for fitted parameters does not apply.

### 2.4.2 Limitations of the $\chi^2$ Approximation

The $\chi^2$ approximation only holds under several assumptions, which are partially violated or hold only asymptotically in our setting. The main limitations are the following:

- For the normal approximation, the standard rule is that the *expected* counts $f'_k$ should be at least 5. This assumption is not always true in our setting, particularly in the tail, where $f'_k$ may be small.
- Agresti and Ryu [1] described the test inversion for Pearson's $\chi^2$ statistic rather than Neyman's. We use Neyman's $\chi^2$ statistic for the convexity reason described in Section 2.3.
- The program is feasible if and only if the minimum of Neyman's $\chi^2$ statistic (obtained by the optimization problem in Section 2.3) is at most $\varepsilon$. Since $\varepsilon$ grows with the degrees of freedom, feasibility is sensitive to the choice of $K$. However, the value of $K$ is not uniquely determined and can be increased by appending up to $C - K$ zero entries to the observed fingerprint. By using the ideas of Section 2.5 and Section 2.6, we obtain feasible optimization problems in practice.

Due to these limitations, the confidence interval construction is best understood as a heuristic rather than a fully derived asymptotic procedure. We show in Section 3 that the resulting confidence intervals empirically achieve near-nominal coverage. The empirical tendency towards over-coverage rather than under-coverage may be explained by two conservative effects: (1) When the expected counts $f'_k$ are small, the distribution of $(f_k - f'_k)^2/f_k$ is right-skewed compared to its asymptotic $\chi^2$ approximation, so a symmetric $\chi^2$ critical value inflates $\varepsilon$. (2) Since each species can only contribute to a single bin, there is a small local correlation. This underdispersion is negligible at small $k$ but may lead to wider intervals.

## 2.5 Estimating Truncation Parameters $C$ and $K$

The explicit modelling of the sampling process depends on the parameters $C \le N$ and $K \le n$, which determine the length of the variable vectors $a$ and $f'$. We follow the strategy of Schröder and Rahmann [38] to determine $C$ and $K$. We determine them in sequence: $K_{\text{obs}}$ is read from $f$, then $C$ is chosen from $K_{\text{obs}}$ via a threshold $t$, then $K$ ($\ge K_{\text{obs}}$) is chosen from $C$ via a threshold $\tau$. Typically, both $C$ and $K$ increase with an increasing sample size.

Let $K_{\text{obs}}$ denote the largest count observed in the sample fingerprint $f$. Assuming that we have a species with abundance $c$ in the population, the probability to observe a count larger than $K_{\text{obs}}$ in our sample is given by $\sum_{k>K_{\text{obs}}} P^{N,n}_{c,k}$. Since no such count was observed, we require that the tail probability $\sum_{k>K_{\text{obs}}} P^{N,n}_{c,k} = 1 - \sum_{k=0}^{K_{\text{obs}}} P^{N,n}_{c,k}$ is small. Given a threshold $t \in (0,1)$, we therefore set

$$C := \min\{c \ge 1 : \sum_{k=0}^{K_{\text{obs}}} P^{N,n}_{c,k} \le t\}\ .$$

Since the optimization program only measures the deviation between the observed and expected fingerprint up to $K$, we must ensure that essentially all the probability mass lies in the considered range. For example, if we observe no species with more than 10 individuals, the entries in $\{f'_k\}_{k>10}$ should also be near zero. For a given $C$ and a threshold $\tau \in (0,1)$, we set

$$K := \min\{\kappa \geq 1 : \sum_{k=0}^{\kappa} P_{C,k}^{N,n} \geq \tau\} .$$

The observed fingerprint is padded with $K - K_{\text{obs}}$ trailing zeros.

The two thresholds play different roles: $t$ caps $C$ from above, so the more conservative (smaller) $t$, the larger $C$ and the more expensive the program; $\tau$ caps how much tail mass of the per-$c$ subsampling distribution we allow to be ignored, with little additional cost when set close to 1. We therefore use $t = 0.8$ and $\tau = 0.99$ as default parameters. The resulting richness estimates are insensitive to the choice of $t$ and $\tau$ as long as both parameters are within a reasonable range (see Appendix B).

## 2.6 Reducing the Problem Size

A challenge arises for samples where both $K$ and $C$ are large, resulting in impractical solving times that exceed the typical few seconds, see Appendix A. RichnEst reduces the program size by truncating the observed fingerprint at a fixed threshold $K_{\max}$ (default 25), only retaining $(f_1, \ldots, f_{K_{\max}})$ and rescaling $n$ and $N$ accordingly [38]. However, this threshold is not very robust to varying input properties. A small threshold reduces the degrees of freedom of the $\chi^2$ distribution, which can render the confidence-interval program infeasible. The extent to which this affects the solvability of the confidence interval is shown in Appendix A.

Instead, we use the rare/abundant fingerprint split proposed by Valiant and Valiant [40], which uses a sparsity criterion that is sensitive to the sample size and population composition rather than relying on a constant $K_{\max}$. If a species is very abundant in the population – say, it makes up half – then its sample frequency will be close to its expected value $n/2$. Thus, we can use the trivial empirical estimate of $N/2$ for this species.

Concretely, an abundant species' sample frequency is tightly concentrated around its expectation, so in the fingerprint $f$ the values $f_i$ for $i$ near that frequency are small; only a few species contribute counts in that neighbourhood. We therefore say that a frequency $k$ is *abundant* if its neighbourhood $f$ is sparse:

$$\sum_{i \in [k - \lceil\sqrt{k}\rceil, k + \lceil\sqrt{k}\rceil]} f_i \;\leq\; 2\sqrt{k} .$$

We remove each *abundant* $f_k$ from the fingerprint, give it an estimated population abundance of $k \cdot N/n$, and reduce $n$ and $N$ by $k \cdot f_k$ and $k \cdot f_k \cdot N/n$, respectively. After solving the reduced program, the empirical entries are added back to recover the full population fingerprint $a$.

This split has been shown to work well in practice [40]. The plug-in is justified because the sparse-neighbourhood criterion identifies precisely those frequencies whose corresponding species are tightly concentrated around their expectation by Le Cam's theorem, so $k \cdot N/n$ is itself a low-variance estimator, and the corresponding entropy contribution $-(k/n)\log(k/n)$ closely tracks the population value $-(c/N)\log(c/N)$.

Another advantage of reducing the fingerprint is that removing sparsely populated bins with small expected counts and few species leaves a compact fingerprint. This fingerprint typically has large enough expected counts to support the normal approximation underlying the Neyman $\chi^2$ statistic.

## 2.7 Entropy Estimation

The optimization problem directly yields an estimate of the population fingerprint $a$. We thus estimate the entropy of a population with upsampled fingerprint $a = (a_1, a_2, \ldots, a_C)$ and population size $N$ as

$$\hat{H}_a = -\sum_{c=1}^{C} a_c \frac{c}{N} \log \frac{c}{N} \,. \tag{10}$$

That is, we plug the estimated population fingerprint instead of the observed sample fingerprint into Equation (1).

To obtain entropy confidence intervals, we replace Equation (4) with Equation (10) and minimize or maximize the optimization program from Section 2.4.1 to obtain the lower and upper bounds of the confidence interval, respectively. Since Equation (10) is linear in $a$, the convexity of the optimization program is conserved.

# 3 Results

We compare DivQuant against the observed richness $S_{\text{obs}}$ and the best-performing extrapolating estimators from the review by Schmitz and Rahmann [36] on simulated and biological data, namely, its predecessor, RichnEst [38] (`https://gitlab.com/rahmannlab/richnest`, commit de571d68), iNEXT [24] (R package, version 3.0.2), and PreSeq [14] (R package, version 4.0.0).

For entropy estimation, we compare DivQuant against the observed sample entropy, given by Equation (1), the asymptotic Miller-Madow bias-corrected estimator [30], the coverage adjusted estimator (CAE) [10], Unseen [40] and the extrapolating entropy estimator available in iNEXT [24]. For Unseen, we use our Python re-implementation of the original MATLAB code. DivQuant, RichnEst and our re-implementation of Unseen use the Gurobi Python API to solve the (linear or quadratic) convex optimization problems [21]. For convex optimization problems, Gurobi yields the global optimum within numerical tolerances. All experiments were run on an Ubuntu (24.04 LTS) server with two AMD EPYC 9534 64-Core Processors, 1.5 TB of 4800-MHz DDR5 memory, and a KIOXIA CD8P SSD. We measured running times and maximum memory usage with `/usr/bin/time -v`.

## 3.1 Simulated Data

We simulate data with different species compositions using the statistical population models described in the review by Schmitz and Rahmann [36]. We set the population size to $N = 10^5$ and the richness to $S = 10^4$. Given a simulated population, we create samples that contain 1% to 90% of the population. We denote the absolute abundance vector of a population by $X = (X_1, \ldots, X_S)$, and use $\alpha$ to denote the appropriate scaling factor such that $\sum_{i=1}^{S} X_i = N$. The simulated species composition models are as follows.

In the *Random* model, we draw $p_i$ for $i = 1, \ldots, S$ from the uniform distribution on $[0.02, 0.98]$ and set $X_i = \alpha \cdot p_i$.

In the *Homogeneous* model, every species $i = 1, \ldots, S$ has the same abundance of $N/S$ (here $10^5/10^4 = 10$).

In the *Negative Binomial* model, we draw a random sample $X'$ of size $S$ from a negative binomial distribution either with parameters $r = 2$ and $p = 0.02$ (model 1), or with $r = 20$ and $p = 0.2$ (model 2). For $i = 1, \ldots, S$, we set $X_i = \alpha \cdot X'_i$.

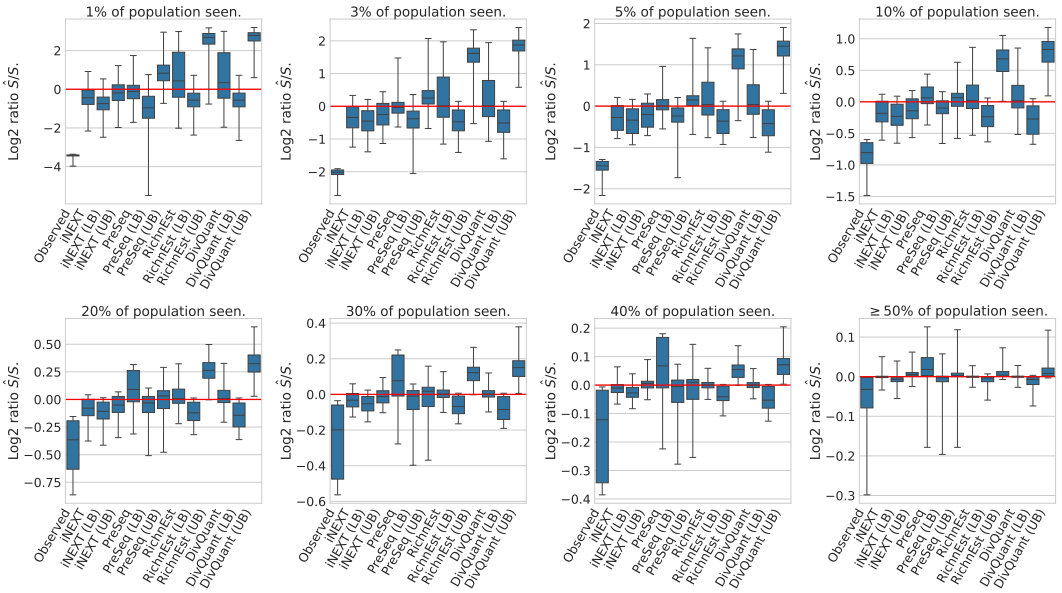



**Figure 2** $\text{Log}_2$ ratios between estimated richness and population richness across different samples from the simulated data. A $\log_2$ ratio of zero corresponds to a perfect correspondence, while $\log_2$ ratios $< 0$ underestimate the true richness and $\log_2$ ratios $> 0$ overestimate the true richness. Each method's point estimate, together with the lower and upper bounds of the computed 95% confidence interval, is shown. DivQuant's intervals are wider than RichnEst's by design; Figure 3 shows that this width translates into well-calibrated coverage, whereas the narrower intervals of competing methods systematically miss the truth.

To create a bimodal composition (*Two Mixture* model), we draw a random sample $X'$ from a negative binomial distribution with parameters $r = 2$ and $p = 0.1$ for one half of the species and with parameters $r = 50$ and $p = 0.2$ for the other half of the species. The absolute abundances are then given according to the description above.

In the *Geometric* model, we draw a sample $X'$ from a geometric distribution with parameter $p = 1/S$ and set $X_i = \alpha \cdot X'_i$ such that $\sum_{i=1}^{S} X_i = N$.

In the *Zipf-Mandelbrot* model, $X_i = \alpha/(i + 10)$ for all $i = 1, \ldots, S$.

### 3.1.1 Richness Estimation

We first compare the richness estimates of DivQuant, RichnEst, iNEXT, and PreSeq on simulated data. Figure 2 shows the $\log_2$ ratio between the estimated and true richness, aggregated across all distribution types, see Appendix C for estimates split by distribution.

As expected, the observed sample richness underestimates the true population richness, and all extrapolating estimators improve as the sampling fraction grows. For small samples ($< 10\%$), PreSeq yields more accurate point estimates compared to iNEXT, RichnEst, and DivQuant. For larger samples ($> 10\%$), both RichnEst and DivQuant are more accurate than PreSeq and iNEXT. The point estimates of RichnEst and DivQuant are similar throughout, but DivQuant's confidence intervals are wider.

While narrower 95% confidence intervals are generally preferred, an interval is only meaningful if it covers the true value at the nominal rate. Neither iNEXT nor PreSeq achieves this. The upper bound of iNEXT is often below the true richness, and PreSeq's bootstrap intervals frequently fail to cover the point estimate.

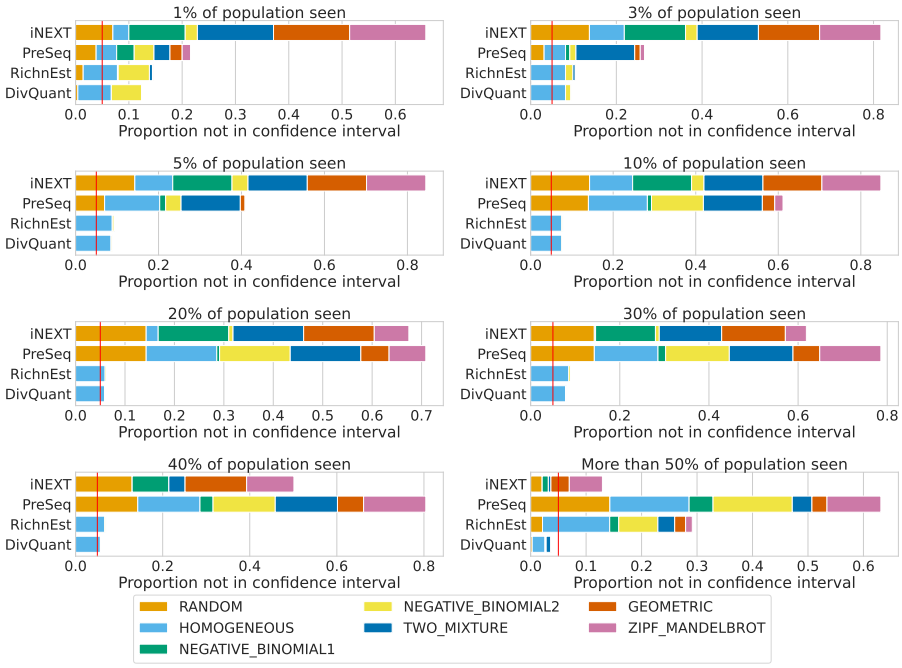



**Figure 3** Fraction of simulated instances where the true richness is not in the 95% confidence interval; split for different sampling fractions. The vertical red line marks the theoretically allowed portion of instances (5%) that are not within the confidence interval. Colours indicate the distribution model from which the failed instances originate.

This is quantified in Figure 3. DivQuant's intervals contain the true richness in essentially all instances, except for the *Homogeneous* and *Negative Binomial* models at small sampling fractions ($\leq 3\%$). For sampling fractions above 50%, DivQuant's intervals are well-calibrated, whereas more than 5% of RichnEst's intervals fail to cover the true richness. For small sampling fractions, fewer than 20% of iNEXT intervals and at most 80% of PreSeq intervals cover the true richness. Even at a sampling fraction of 40%, only about half of iNEXT's intervals and roughly 20% of PreSeq's intervals are correct.

In summary, of the four methods compared, DivQuant is the only one whose confidence intervals closely approach the nominal 95% level across the simulated samples. By contrast, iNEXT's and PreSeq's confidence intervals seldom contain the true richness.

### 3.1.2 Entropy Estimation

We next compare the estimated entropy on simulated data. Since iNEXT computes entropy in base 10, we report all entropies using $\log_{10}$. DivQuant supports $\log_2, \log_{10}$ and $\log_e$.

Figure 4 shows that the observed sample entropy generally underestimates the true entropy. For larger sampling fractions, the asymptotic estimators (Miller-Madow, CAE, Unseen), and the extrapolating estimator, iNEXT, begin to overestimate the true entropy. DivQuant yields the most accurate entropy estimates for sampling fractions $> 3\%$.

Miller-Madow, CAE and Unseen do not provide confidence intervals. The confidence intervals of iNEXT do not contain the true entropy for a large portion of tested instances, see Figure 4 and Figure 5. By contrast, DivQuant's confidence intervals contain the true entropy in a larger proportion of cases. For a small sampling fraction, more than 5% of instances are not within the confidence interval; for a larger sampling fraction, only estimates for the *Homogeneous* model are not within the confidence interval.

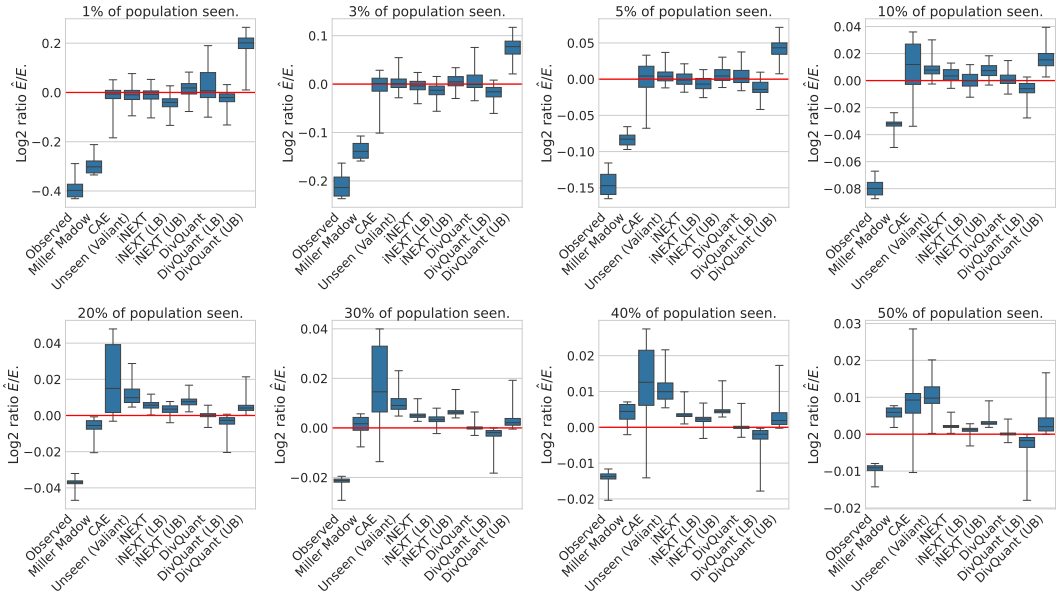



**Figure 4** $\text{Log}_2$ ratio between estimated entropy and population entropy across different samples of the simulated data. For iNEXT and DivQuant, LB and UB denote the confidence intervals. A $\log_2$ ratio of zero corresponds to a perfect correspondence, while $\log_2$ ratios $< 0$ underestimate the true entropy and $\log_2$ ratios $> 0$ overestimate the true entropy.

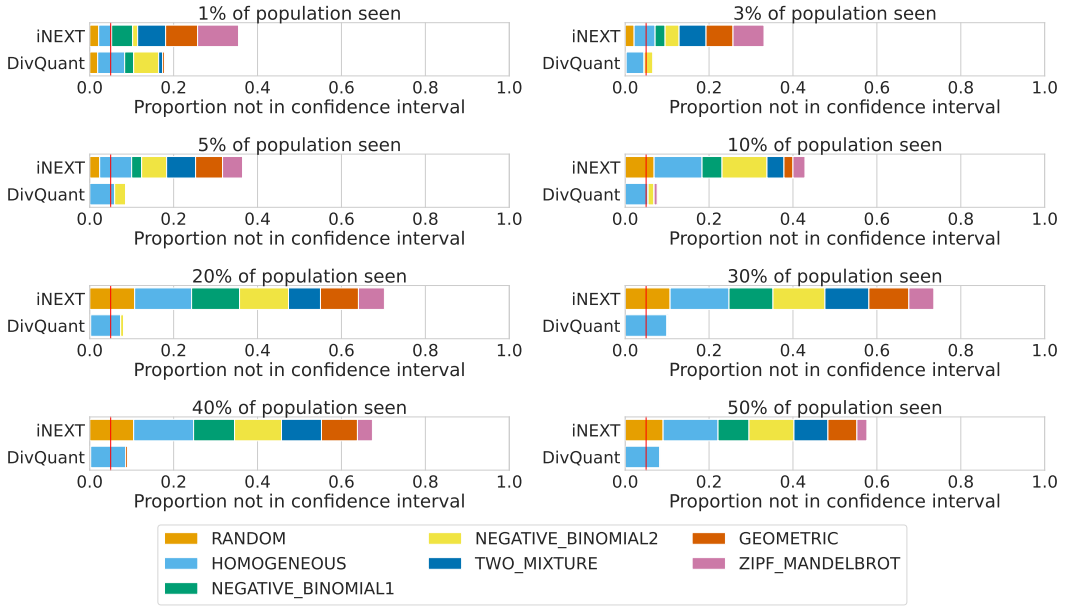



**Figure 5** Fraction of simulated instances where the true entropy is not in the 95% confidence interval; split for different sampling fractions. The vertical red line marks the theoretically allowed portion of instances (5%) that are not within the confidence interval. Colours indicate the distribution model from which the failed instances originate.

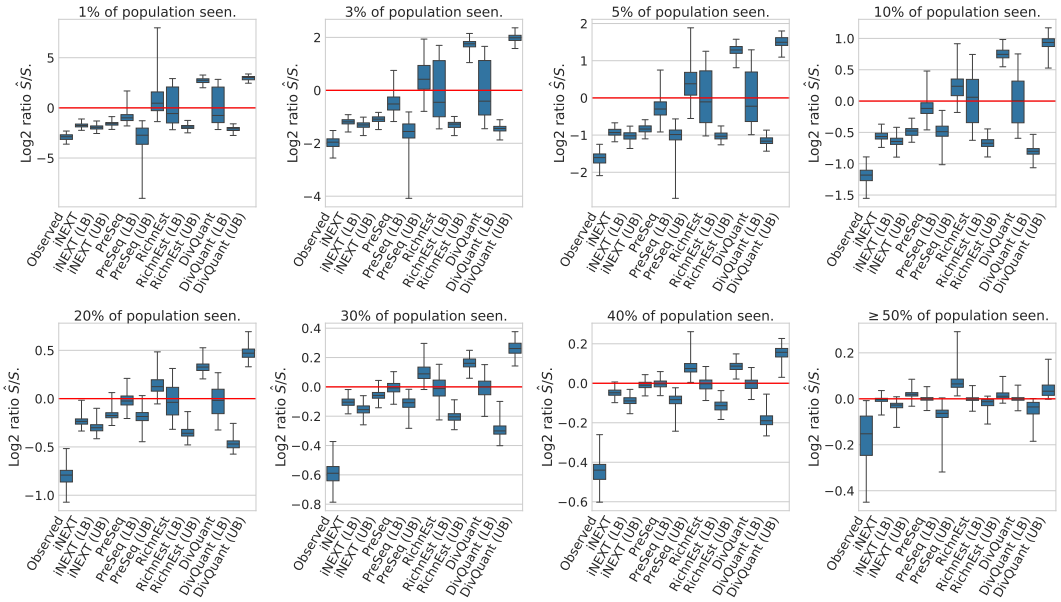



**Figure 6** $\log_2$ ratio between estimated richness and population richness for the *Tara Oceans* microbial taxonomic profiles across different sampling fractions. A $\log_2$ ratio of zero corresponds to a perfect correspondence, while $\log_2$ ratios $< 0$ underestimate the true richness and $\log_2$ ratios $> 0$ overestimate the true richness. For all methods, the point estimate and the lower and upper bounds of the computed 95% confidence interval are shown.

## 3.2 Microbiome Richness

We obtained the taxonomic profiles from the *Tara Oceans* consortium [5], which sequenced plankton samples across all major oceanic regions. From the operational taxonomic unit (OTU) count tables of the 139 different samples (here treated as the "population", available at `ocean-microbiome.embl.de/companion.html`), we drew random subsamples that contain $\rho = 0.01, 0.03, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5$ of the complete sample. For each sample (population), we compared the estimated richness on the subsample with the richness of the complete sample. A fair comparison with asymptotic estimators is not possible because the full population is unknown. We report entropy results for completeness in Appendix D.

Figure 6 shows that, similarly to the results on simulated data, all methods yield more accurate richness estimates and smaller confidence intervals as the sampling fraction increases. DivQuant has larger confidence intervals compared to PreSeq and RichnEst. However, both PreSeq and RichnEst have a larger portion of instances (6% and 10%) where the correct richness is not contained within the confidence interval (see Figure 7A). The upper bound of iNEXT underestimates the true richness in more than half of the instances.

## 3.3 Duplicate Rate Estimation

We downloaded the scRNA-seq data generated by *10X Genomics* from `https://cg.10xgenomics.com/samples/cell-exp/3.0.2/5k_pbmc_v3/5k_pbmc_v3_fastqs.tar`. Each read in this data set contains a sequence tag of length 28 that should be unique for each distinct molecule. Given the number $S$ of distinct sequence tags in an experiment with $N$ reads, the duplicate rate is $(N - S)/N$. In practice, one is often interested in the duplicate rate at an increased sequencing depth (for example, sequencing $2\times$ to $10\times$ as many reads) in order to determine whether further sequencing would result in new molecules being sequenced. We counted the sequence tags using the $k$-mer counter Hackgap [45].

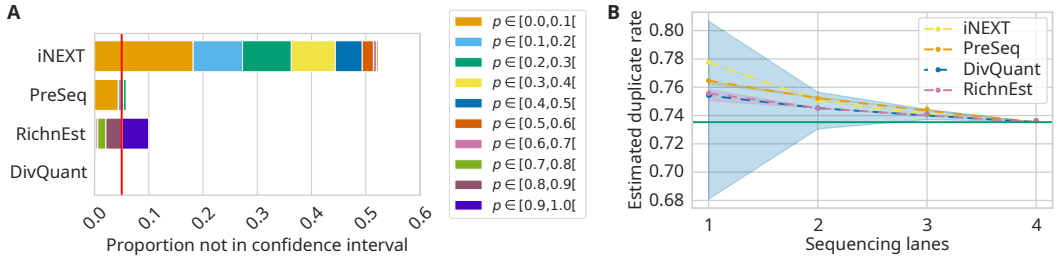



**Figure 7** **A** Fraction of *Tara Oceans* instances where the true richness is not in the 95% confidence interval. Colours indicate the sampling fraction. **B** Confidence interval for estimated duplicate rates for the *10X Genomics* scRNA-seq data. Lanes 1 to 4 cumulatively contain $n = 96\,039\,314$, 192 665 228, 288 186 226, and 382 950 653 sequences, with observed richnesses $S_{\text{obs}} = 54\,405\,046$, 77 172 440, 90 772 183, 101 353 538, respectively. The horizontal green line corresponds to the duplicate rate in the union of all 4 sequencing lanes.

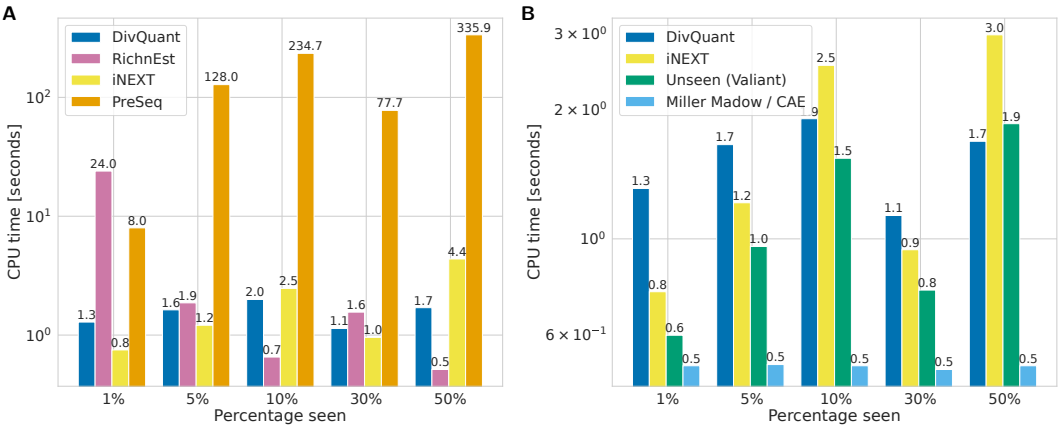



**Figure 8** Running time in CPU seconds for the *Tara Oceans* sample `TARA_004_DCM_0.22-1.6` ($N = 143\,169$). Running times are shown for (**A**) richness estimators and (**B**) entropy estimators. The running times (y-axis, log scale) are averaged over three runs for five different sampling fractions (x-axis).

To obtain a ground-truth duplicate rate at increased depth, we pool the reads from all four lanes and use the duplicate rate of the pool (horizontal green line in Figure 7B) as the target. By adding more individual lanes to a sample, we obtain samples of increasing size and smaller confidence intervals. The true target value should be inside the confidence interval, and the point estimate should be close to the target. Figure 7B shows the estimated duplicate rate. Consistent with previous results, DivQuant has the largest confidence intervals but is the only method where the true duplicate rate lies within the confidence interval. All other methods overestimate the duplicate rate, i.e., underestimate the true richness (not shown).

## 3.4 Running Time and Memory Requirements

DivQuant estimates both richness and entropy within a few seconds on both simulated and real data. Across all *Tara Oceans* samples (and sampling fractions), the maximum CPU time is 17 s, and the maximum memory usage is 206 MB. Figure 8 compares the running times of the methods (both richness and entropy estimators) for different subsampling fractions for the `TARA_004_DCM_0.22-1.6` sample. All reported times are averaged over three runs.

For richness estimation (see Figure 8A), RichnEst takes a few seconds for all subsampling fractions except 1%, where it takes 24 s. PreSeq is the slowest method and scales poorly, from about 8 s at 1% to roughly 5 min 36 s at 50%. Both DivQuant and iNEXT require below 5 seconds. As expected, the Miller-Madow and CAE estimators are the fastest entropy estimators (see Figure 8B), since they have closed-form expressions. On this dataset, DivQuant, iNEXT, and Unseen need below 3 seconds at all subsampling fractions.

## 4 Discussion and Conclusion

We have presented an optimization approach, called DivQuant, to estimate richness and entropy of a population when given access to only a small sample at a known sampling fraction. DivQuant provides both point estimates and confidence intervals.

Our experiments indicate that our new approach computes more accurate richness and entropy confidence intervals compared to other state-of-the-art methods. For sampling fractions $< 10\%$, the richness confidence intervals are relatively broad, but the true solution falls within the theoretical 95% confidence interval for nearly all the tested instances. The main exception is the homogeneous distribution: when all species share the same abundance, the sample fingerprint concentrates near a single value $k \approx n/S$, leaving the $\chi^2$ statistic with essentially one informative bin and the test inversion with little leverage. This affects richness coverage at small sampling fractions and entropy coverage at larger sampling fractions; a tailored treatment of this corner case is left for future work. In contrast, more than 5% of the confidence intervals computed by RichnEst, iNEXT, and PreSeq fail to cover the true richness. For entropy, plugging the optimal population fingerprint into Shannon's entropy formula produces more accurate point estimates than the asymptotic estimators (Miller-Madow, CAE, Unseen) and the extrapolating estimator iNEXT. In addition, the entropy confidence intervals of DivQuant contain the true solution in more instances compared to iNEXT.

DivQuant requires the population size $N$ to be known. When $N$ is only known approximately, DivQuant can be run over a range of plausible $N$ values and the minimum lower-bound richness across that range reported as a conservative estimate; because larger $N$ adds unseen species rather than restructuring the observed fingerprint, this lower bound tends to saturate.

In future work, we aim to further tighten the statistical analysis for richness estimation and to increase the accuracy of entropy confidence intervals. DivQuant currently relies on a free academic Gurobi license (and its Python API). Evaluating open-source QP backends (e.g., OSQP, Ipopt, or CVXPY's SCS/clarabel) is planned as future work.

In summary, DivQuant is well-suited to estimate richness and entropy in the extrapolating setting, where the population size (or a reliable estimate of it) is known.

## 5 Data and Code Availability

DivQuant and all Snakemake [31] workflows to reproduce the experiments are available at `https://gitlab.com/rahmannlab/divquant` (commit `2baab79d`). The optimization problems are solved using a free academic Gurobi licence [21].

## References

1 Alan Agresti and Euijung Ryu. Pseudo-score confidence intervals for parameters in discrete statistical models. *Biometrika*, 97(1):215–222, 2010. doi:10.1093/biomet/asp074.

2 Steve Baker and Robert D. Cousins. Clarification of the use of CHI-square and likelihood functions in fits to histograms. *Nuclear Instruments and Methods in Physics Research*, 221(2):437–442, 1984. doi:10.1016/0167-5087(84)90016-4.

3 Kathryn Barger and John Bunge. Objective Bayesian estimation for the number of species. *Bayesian Analysis*, 5(4):765–785, 2010. doi:10.1214/10-BA527.

4 Joseph Berkson. Minimum Chi-Square, not Maximum Likelihood! *The Annals of Statistics*, 8(3):457–487, 1980. doi:10.1214/aos/1176345003.

5 Peer Bork, Chris Bowler, Colomban de Vargas, Gabriel Gorsky, Eric Karsenti, and Patrick Wincker. Tara Oceans studies plankton at planetary scale. *Science*, 348(6237):873–873, 2015. doi:10.1126/science.aac5605.

6 Hendriek C. Boshuizen and Dennis E. te Beest. Pitfalls in the statistical analysis of microbiome amplicon sequencing data. *Molecular Ecology Resources*, 23(3):539–548, 2023. doi:10.1111/1755-0998.13730.

7 Kenneth P. Burnham and Scott W. Overton. Estimation of the size of a closed population when capture probabilities vary among animals. *Biometrika*, 65(3):625–633, 1978. doi:10.1093/biomet/65.3.625.

8 Anne Chao. Nonparametric Estimation of the Number of Classes in a Population. *Scandinavian Journal of Statistics*, 11(4):265–270, 1984. URL: https://www.jstor.org/stable/4615964.

9 Anne Chao, Nicholas J. Gotelli, T. C. Hsieh, Elizabeth L. Sander, K. H. Ma, Robert K. Colwell, and Aaron M. Ellison. Rarefaction and extrapolation with Hill numbers: a framework for sampling and estimation in species diversity studies. *Ecological Monographs*, 84(1):45–67, 2014. doi:10.1890/13-0133.1.

10 Anne Chao and Tsung-Jen Shen. Nonparametric estimation of Shannon's index of diversity when there are unseen species in sample. *Environmental and Ecological Statistics*, 10(4):429–443, 2003. doi:10.1023/A:1026096204727.

11 Chun-Huo Chiu. A more reliable species richness estimator based on the Gamma–Poisson model. *PeerJ*, 11:e14540, 2023. doi:10.7717/peerj.14540.

12 Chun-Huo Chiu and Anne Chao. Estimating and comparing microbial diversity in the presence of sequencing errors. *PeerJ*, 4:e1634, 2016. doi:10.7717/peerj.1634.

13 Robert K. Colwell, Anne Chao, Nicholas J. Gotelli, Shang-Yi Lin, Chang Xuan Mao, Robin L. Chazdon, and John T. Longino. Models and estimators linking individual-based and sample-based rarefaction, extrapolation and comparison of assemblages. *Journal of Plant Ecology*, 5(1):3–21, 2012. doi:10.1093/jpe/rtr044.

14 Timothy Daley and Andrew D. Smith. Predicting the molecular complexity of sequencing libraries. *Nature Methods*, 10(4):325–327, April 2013. doi:10.1038/nmeth.2375.

15 Ian A. Dickie. Insidious effects of sequencing errors on perceived diversity in molecular surveys. *New Phytologist*, 188(4):916–918, 2010. doi:10.1111/j.1469-8137.2010.03473.x.

16 Ronald A. Fisher, Alexander S. Corbet, and Carrington B. Williams. The relation between the number of species and the number of individuals in a random sample of an animal population. *Journal of Animal Ecology*, 12(1):42–58, 1943. doi:10.2307/1411.

17 Nikolay Gagunashvili. Chi-square tests for comparing weighted histograms. *Nuclear Instruments and Methods in Physics Research*, 614(2):287–296, 2010. doi:10.1016/j.nima.2009.12.037.

18 Kate L. Gibson, Yu-Chang Wu, Yvonne Barnett, Orla Duggan, Robert Vaughan, Elli Kondeatis, Bengt-Olof Nilsson, Anders Wikby, David Kipling, and Deborah K. Dunn-Walters. B-cell diversity decreases in old age and is correlated with poor health status. *Aging Cell*, 8(1):18–25, 2009. doi:10.1111/j.1474-9726.2008.00443.x.

19 Irving J. Good and George H. Toulmin. The number of new species, and the increase in population coverage, when a sample is increased. *Biometrika*, 43(1):45–63, 1956. doi:10.1093/biomet/43.1-2.45.

20 François Grin and Guillaume Fürst. Measuring linguistic diversity: A multi-level metric. *Social Indicators Research*, 164(2):601–621, 2022. doi:10.1007/s11205-022-02934-5.

21 Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2026. URL: https://www.gurobi.com.

22 T. Hauschild and Michael Jentschel. Comparison of maximum likelihood estimation and chi-square statistics applied to counting experiments. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 457(1):384–401, 2001. doi:10.1016/S0168-9002(00)00756-7.

23 Yili Hong. On computing the distribution function for the poisson binomial distribution. *Computational Statistics & Data Analysis*, 59:41–51, 2013. doi:10.1016/j.csda.2012.10.006.

24 T. C. Hsieh, K. H. Ma, and Anne Chao. iNEXT: an R package for rarefaction and extrapolation of species diversity (Hill numbers). *Methods in Ecology and Evolution*, 7(12):1451–1456, 2016. doi:10.1111/2041-210X.12613.

25 T.C. Hsieh and Anne Chao. Rarefaction and extrapolation: Making fair comparison of abundance-sensitive phylogenetic diversity among multiple assemblages. *Systematic Biology*, 66(1):100–111, 2017. doi:10.1093/sysbio/syw073.

26 Xiangpan Ji, Wenqiang Gu, Xin Qian, Hanyu Wei, and Chao Zhang. Combined Neyman–Pearson chi-square: An improved approximation to the Poisson-likelihood chi-square. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 961:163677, 2020. doi:10.1016/j.nima.2020.163677.

27 Krisana Lanumteang and Dankmar Böhning. An extension of Chao's estimator of population size based on the first three capture frequency counts. *Computational Statistics & Data Analysis*, 55(7):2302–2311, 2011. doi:10.1016/j.csda.2011.01.017.

28 Daniel J. Laydon, Charles R. M. Bangham, and Becca Asquith. Estimating T-cell repertoire diversity: limitations of classical estimators and a new approach. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 370(1675):20140291, 2015. doi:10.1098/rstb.2014.0291.

29 Lucien Le Cam. An approximation theorem for the poisson binomial distribution. *Pacific Journal of Mathematics*, 10(4):1181–1197, 1960. URL: https://msp.org/pjm/1960/10-4/p11.xhtml.

30 George Miller. Note on the bias of information estimates. *Information theory in psychology : Problems and methods*, 1955.

31 Felix Mölder, Kim P. Jablonski, Brice Letcher, Michael B. Hall, Christopher H. Tomkins-Tinch, Vanessa Sochat, Jan Forster, Soohyun Lee, Sven O. Twardziok, Alexander Kanitz, Andreas Wilm, Manuel Holtgrewe, Sven Rahmann, Sven Nahnsen, and Johannes Köster. Sustainable data analysis with Snakemake. *F1000Research*, 2021. doi:10.12688/f1000research.29032.1.

32 Jerzy Neyman. Contribution to the theory of the $\chi^2$ test. *Proceedings of the Berkeley Symposium on Mathematical Statistics and Probability*, pages 239–273, 1949. URL: https://projecteuclid.org/ebook/Download?urlid=bsmsp/1166219208&isFullBook=false.

33 Liam Paninski. Estimation of Entropy and Mutual Information. *Neural Computation*, 15(6):1191–1253, 2003. doi:10.1162/089976603321780272.

34 Nicla Porciello, Ornella Franzese, Lorenzo D'Ambrosio, Belinda Palermo, and Paola Nisticò. T-cell repertoire diversity: friend or foe for protective antitumor response? *Journal of Experimental & Clinical Cancer Research*, 41(1):356, 2022. doi:10.1186/s13046-022-02566-0.

35 Patrick D. Schloss. Rarefaction is currently the best approach to control for uneven sequencing effort in amplicon sequence analyses. *mSphere*, 9(2):e00354–23, 2024. doi:10.1128/msphere.00354-23.

36 Johanna E. Schmitz and Sven Rahmann. A comprehensive review and evaluation of species richness estimation. *Briefings in Bioinformatics*, 26(2):bbaf158, 2025. doi:10.1093/bib/bbaf158.

37 Johanna Elena Schmitz and Sven Rahmann. DivQuant. Software, version 0.1., swhId: swh:1:dir:1ca96b0e0bbf05c5e7afa9dee8101fd46e1246b5 (visited on 2026-08-12). URL: https://gitlab.com/rahmannlab/divquant, doi:10.4230/artifacts.27620.

38 Christopher Schröder and Sven Rahmann. Efficient duplicate rate estimation from subsamples of sequencing libraries. *PeerJ PrePrints*, 3:e1298v2, September 2015. doi:10.7287/peerj.preprints.1298v2.

39 Claude E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948. doi:10.1002/j.1538-7305.1948.tb01338.x.

40 Gregory Valiant and Paul Valiant. Estimating the Unseen: Improved Estimators for Entropy and Other Properties. *Journal of the ACM*, 64(6):37:1–37:41, October 2017. doi:10.1145/3125643.

41 Cameron Wagg, Klaus Schlaeppi, Samiran Banerjee, Eiko E. Kuramae, and Marcel G. A. van der Heijden. Fungal-bacterial diversity and microbiome complexity predict ecosystem functioning. *Nature Communications*, 10(1):4841, 2019. doi:10.1038/s41467-019-12798-y.

42 Pengyang Wang, Donghao Wu, Yongjia Wang, Zhiqiang Shen, Buhang Li, Zufei Shu, and Chengjin Chu. Tree species richness suppresses red imported fire ant invasion in a subtropical plantation forest. *Journal of Applied Ecology*, 61(11):2751–2761, 2024. doi:10.1111/1365-2664.14786.

43 Amy Willis and John Bunge. Estimating diversity via frequency ratios. *Biometrics*, 71(4):1042–1049, 2015. doi:10.1111/biom.12332.

44 Amy D. Willis. Rarefaction, Alpha Diversity, and Statistics. *Frontiers in Microbiology*, 10, 2019. URL: https://www.frontiersin.org/articles/10.3389/fmicb.2019.02407.

45 Jens Zentgraf and Sven Rahmann. Fast Gapped $k$-mer Counting with Subdivided Multi-Way Bucketed Cuckoo Hash Tables. In Christina Boucher and Sven Rahmann, editors, *22nd International Workshop on Algorithms in Bioinformatics (WABI 2022)*, volume 242 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.WABI.2022.12.


## A Effect of Reducing the Problem Size

We reduce the size of the optimization problem by splitting the observed fingerprint into two parts. The first part (low abundances) is used for the upsampling process, and the second part (high abundances) is upsampled by multiplying the frequency count by $N/n$.

**Effect on running time.** Reducing the problem size decreases the running time for large samples (Figure 9A) for two reasons. First, we have to compute $C \times K$ hypergeometric probabilities to compute the expected fingerprints. Second, the optimization problem has more variables and can thus be harder to solve.

**Effect on richness estimates.** Since we only remove abundant species, whose population frequencies are concentrated around their up-scaled sample frequencies, both versions result in similar point estimates, as demonstrated for one microbial sample in Table 2. Since we have a larger $\chi^2$ error bound for larger $K$, the confidence interval is larger using the complete

**Table 2** Richness estimates for DivQuant with (standard) and without (complete) reducing the problem size on the *Tara Oceans* sample `TARA_004_DCM_0.22-1.6` ($S = 4398, N = 143\,169$). LB and UB denote the lower and upper bounds of the 95% confidence interval.

| Estimator | $n/N$ | 1% | 5% | 10% | 30% | 50% |
|---|---|---|---|---|---|---|
| point | | 1810 | 7322 | 4476 | 4417 | 4408 |
| point (complete) | | 1837 | 7388 | 4527 | 4458 | 4438 |
| LB | | 1189 | 2084 | 2771 | 3751 | 4155 |
| LB (complete) | | 994 | 1816 | 2312 | 3178 | 3622 |
| UB | | 41609 | 12664 | 8256 | 5440 | 4634 |
| UB (complete) | | 53818 | 19706 | 13690 | 7973 | 6126 |

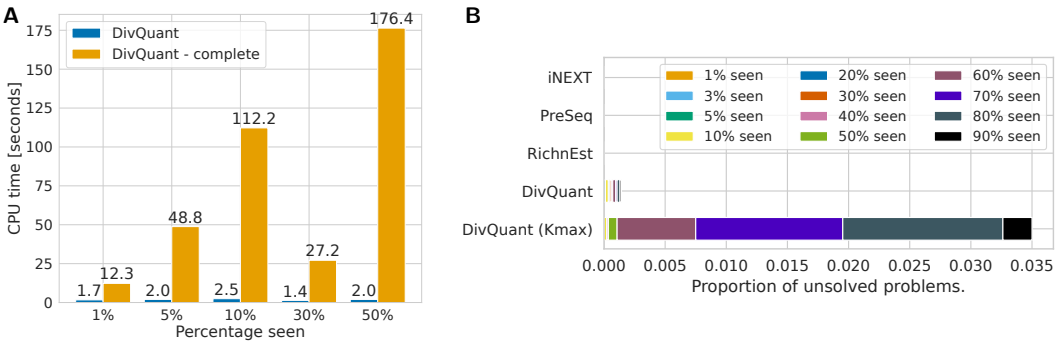



**Figure 9 A** Running time in CPU seconds (y-axis) for different sampling fractions (x-axis) for the *Tara Oceans* sample `TARA_004_DCM_0.22-1.6` ($N = 143\,169$). Running times are shown for DivQuant with (standard) and without (complete) reducing the problem size. The running times are averaged over three runs. **B** Fraction of simulated instances for which no confidence interval could be computed. DivQuant uses the rare/abundant fingerprint split, while DivQuant ($K_{\max}$) truncates at the same fixed threshold of 25 as RichnEst. The splitting scheme is more robust and leaves fewer instances without a feasible confidence-interval program.

fingerprint compared to using only the rare part of the fingerprint to upsample the population. Hence, the wider intervals at small sampling fractions in Figure 2 are intrinsic to the $\chi^2$ inversion when the sample carries little information about the unseen mass, rather than an artefact of the rare/abundant split.

**Effect on the fraction of solved instances.** Figure 9B shows how problem reduction affects the feasibility of the confidence-interval program: Using a fixed threshold (truncation at $K_{\max} = 25$, as in RichnEst [38]) leaves substantially more instances without a feasible solution than the rare/abundant split used in DivQuant.

## B Choosing Thresholds $t$ and $\tau$ to Estimate the Problem Size

Figure 10 shows the mean $\log_2$ ratio between the estimated and true richness for different parameter combinations of $t$ and $\tau$, which determine $C$ and $K$. Figure 11 shows the corresponding standard deviation for the point estimates. DivQuant's point estimates have a small positive bias, in particular for small sampling fractions. The variation of the point estimate decreases for increasing sample sizes. As expected, the lower and upper bounds yield negative and positive $\log_2$ ratios, respectively. The dependence on $t$ and $\tau$ is negligible within a reasonable parameter range. For example, the point estimates are slightly worse if $t \geq 0.8$ and simultaneously $\tau \leq 0.7$ at the smallest sampling fraction, but for all other combinations, the mean and standard deviation are almost identical.

## C Results for Different Population Models

Figures 12 and 13 show the richness and entropy estimates for the different simulation models for a sampling fraction of 10%. Other sampling fractions are not shown due to the limited space. Except for the *Homogeneous* and *Negative Binomial 2* models, iNEXT consistently underestimates the true richness. The performance of PreSeq depends on the population type, e.g., the results are more accurate for a *Geometric* model and less accurate for a *Homogeneous* model. The performance of RichnEst and DivQuant is similar across models.

The Miller-Madow estimator consistently underestimates the true entropy, while Unseen tends to overestimate the true entropy. The CAE estimator overestimates the entropy for most models, but it underestimates the entropy for distribution types with many rare species, i.e., for the *Geometric* and *Zipf-Mandelbrot* models. The performance of iNEXT and DivQuant varies for different models. E.g., for a *Homogeneous* model, iNEXT overestimates entropy (both point estimate and confidence interval), while it provides accurate estimates for the *Geometric* model. The performance of DivQuant is similar across models, but the confidence intervals are larger for the *Zipf-Mandelbrot* model compared to the other models.

## D Estimated Entropy on the Microbiome Data

We estimated the entropy for the data discussed in Section 3.2. Since we do not have access to the whole population, we consider the complete sample as the population. Thus, the estimates of the asymptotic estimators (Miller-Madow, CAE, and Unseen) may deviate from the *true* entropy. Since iNEXT and DivQuant are extrapolating estimators, their entropy estimates should be close to the *true* entropy.

The entropy estimates are shown in Figure 14. For all sampling fractions, DivQuant provides the most accurate entropy estimates. In contrast, the observed sample entropy, all asymptotic entropy estimates, and iNEXT underestimate the true entropy for small sampling fractions and start to overestimate the entropy for larger sampling fractions. On the microbiome data, the entropy confidence intervals of DivQuant are accurate, i.e., DivQuant's lower bound is smaller than the true entropy and DivQuant's upper bound is larger.

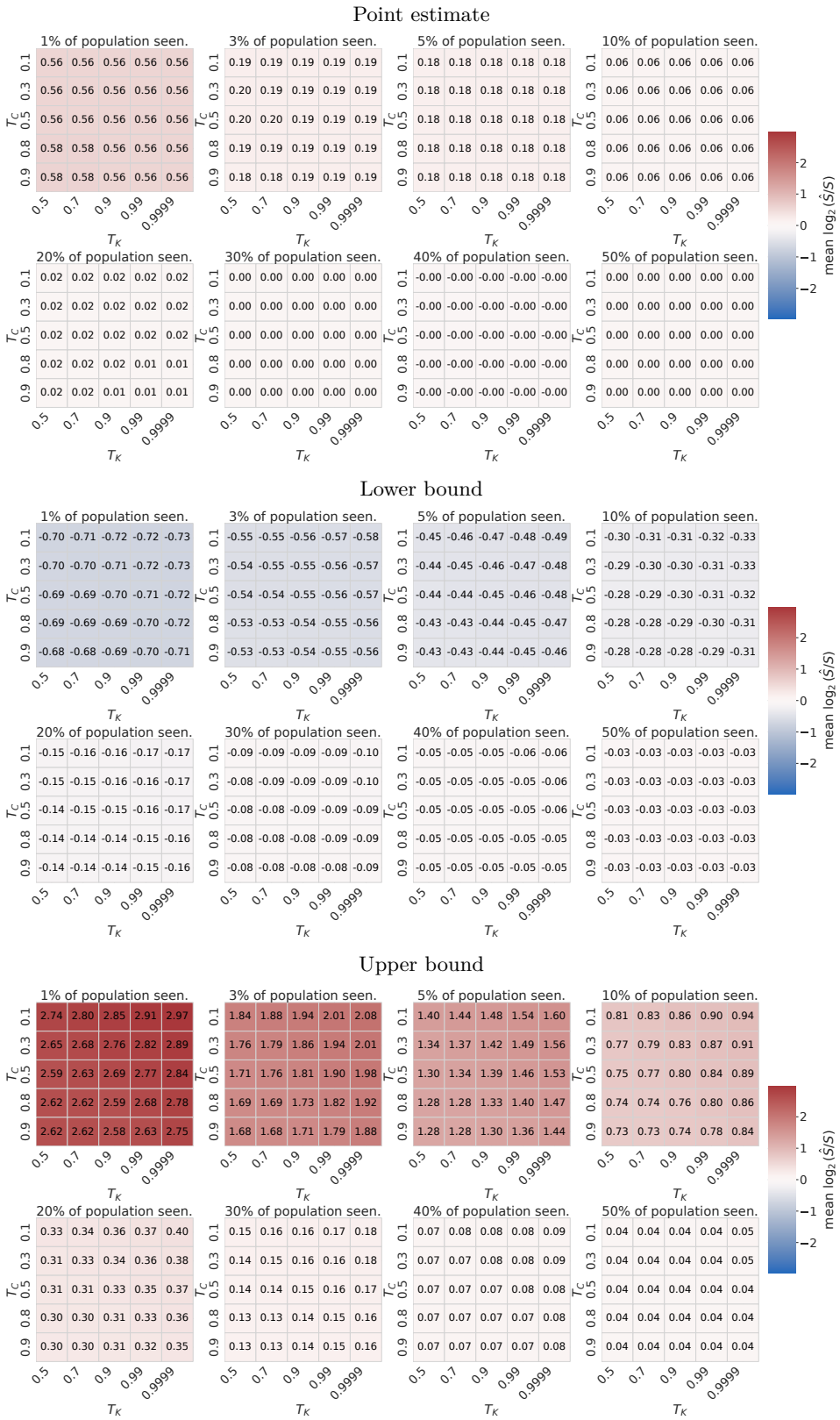



**Figure 10** Mean $\log_2$ ratios of the estimated (point estimate, lower, and upper bounds) and true richness for different parameter combinations of $t$ and $\tau$. A $\log_2$ ratio of zero is best.

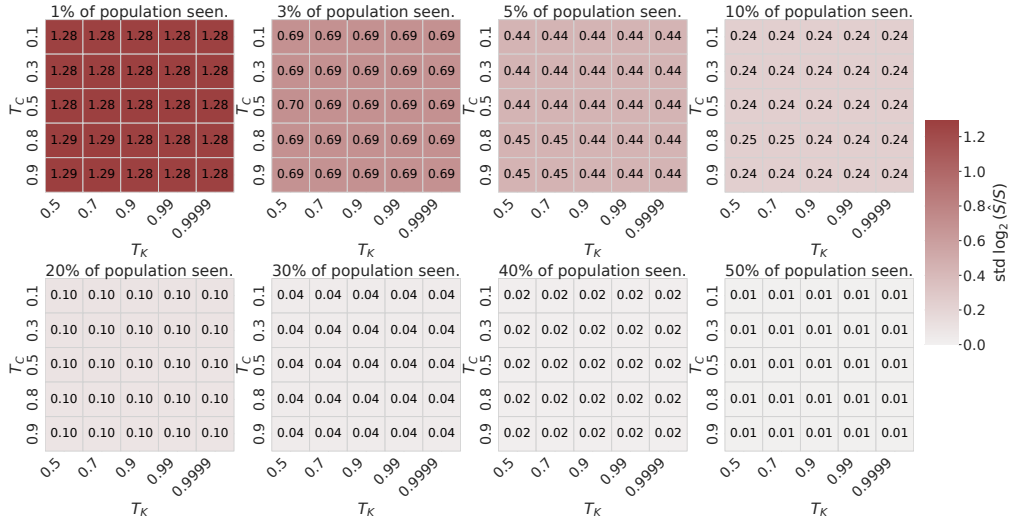



**Figure 11** Standard deviation of the $\log_2$ ratios between the estimated (point estimate, lower and upper bound) and true richness for different parameter combinations of $t$ and $\tau$. Smaller values (close to zero) are better.

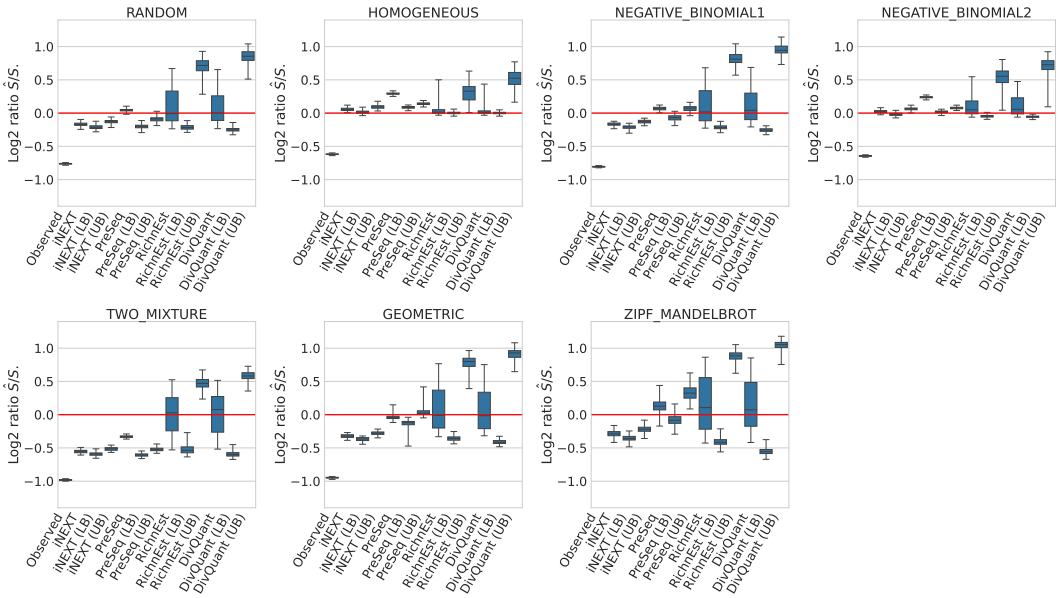



**Figure 12** $\text{Log}_2$ ratios between estimated richness and population richness for a sampling fraction of 10% for different population models. A $\log_2$ ratio of zero corresponds to a perfect correspondence, while $\log_2$ ratios $< 0$ underestimate the true richness and $\log_2$ ratios $> 0$ overestimate the true richness. Each method's point estimate, together with the lower and upper bounds of the computed 95% confidence interval, is shown.

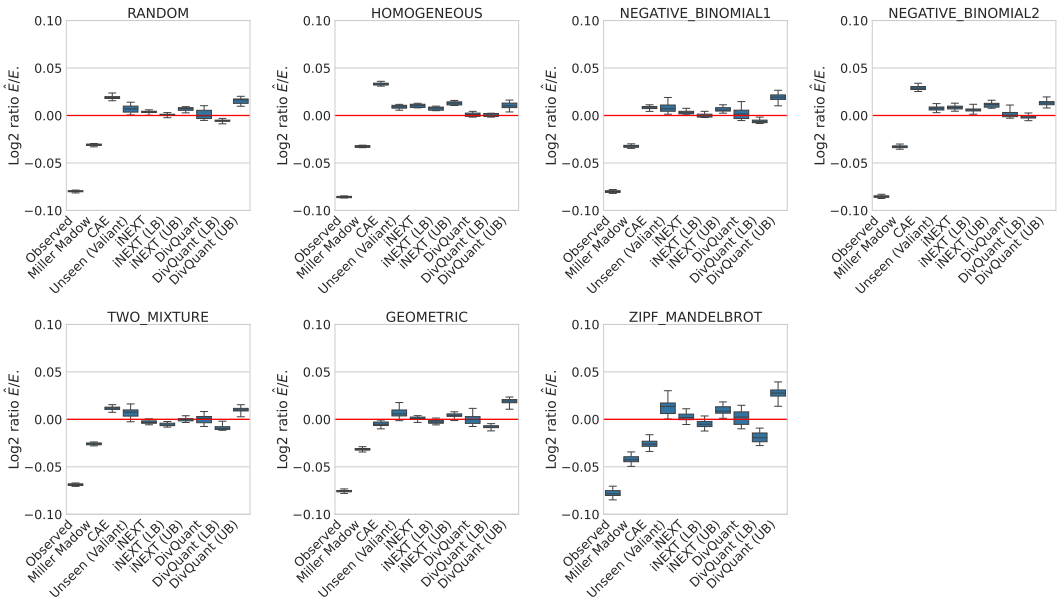



**Figure 13** $\text{Log}_2$ ratios between estimated entropy and population richness for a sampling fraction of 10% for different population models. A $\log_2$ ratio of zero corresponds to a perfect correspondence, while $\log_2$ ratios $< 0$ underestimate the true entropy and $\log_2$ ratios $> 0$ overestimate the true entropy. LB and UB denote the confidence intervals computed by iNEXT and DivQuant.

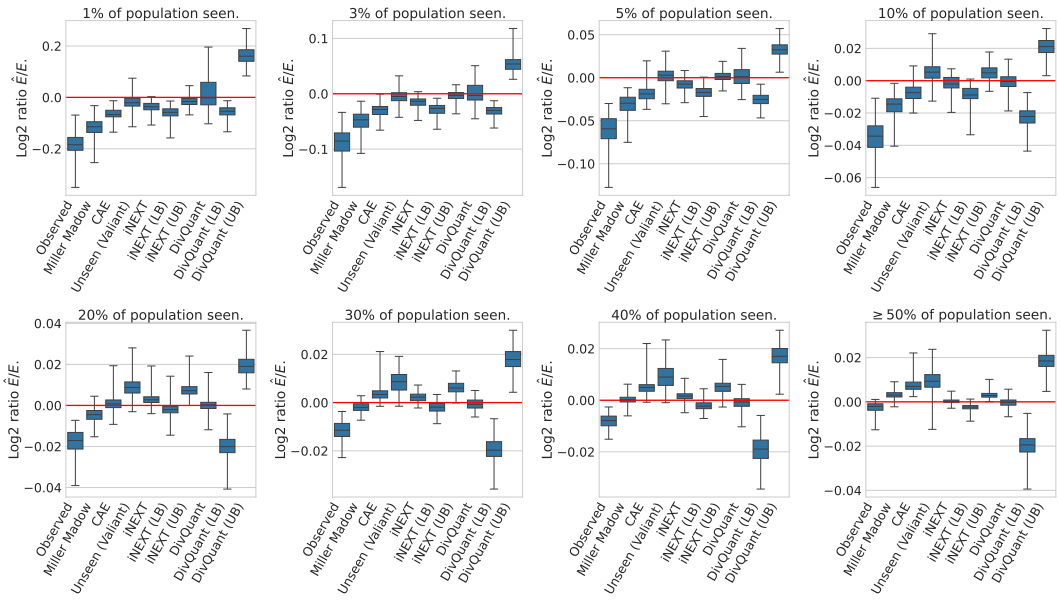



**Figure 14** $\text{Log}_2$ ratios between estimated entropy and true "population" entropy on the *Tara Oceans* microbial taxonomic profiles for different sampling fractions. A $\log_2$ ratio of zero corresponds to a perfect correspondence, while $\log_2$ ratios $< 0$ underestimate the true entropy and $\log_2$ ratios $> 0$ overestimate the true entropy.

# Discriminative Learning of Substitution Matrices and Gap Penalties for Pairwise Alignment of Biological Sequences

**Michał Aleksander Ciach**[1]
Department of Applied Biomedical Science, Faculty of Health Sciences, University of Malta, Msida, Malta

**Elissavet Zacharopoulou**
Department of Applied Biomedical Science, Faculty of Health Sciences, University of Malta, Msida, Malta

**Michał Piotr Startek**
Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland

**Błażej Miasojedow**
Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland

**Panagiotis Alexiou**
Department of Applied Biomedical Science, Faculty of Health Sciences, University of Malta, Msida, Malta

## Abstract

Pairwise alignment scores are used to classify pairs of sequences in many areas of bioinformatics, including homology search, predicting interactions, or read mapping. The relative scores of different pairs strongly depend on the choice of a substitution matrix and gap penalties. However, current approaches for the estimation of these parameters typically describe patterns observed in a collection of ground-truth alignments instead of optimizing specifically for the task of classification. In this work, we present DiscrimAlign, a statistical model for discriminative learning of substitution matrices and gap penalties from a dataset of positive and negative pairs of unaligned DNA or amino acid sequences. The model links the alignment score of a sequence pair with the associated binary label through a logistic function and learns the parameters by likelihood maximization. We analyze theoretical properties of the model, derive and implement a learning procedure, study its performance in simulated experiments, and apply it to predict microRNA-target interactions. We show that sequence alignment with discriminative substitution matrices and gap penalties predicts the interactions comparably to black-box neural network classifiers while being more interpretable. An implementation of the model and reproducibility workflows are available at `https://github.com/BioGeMT/DiscrimAlign`.

**2012 ACM Subject Classification** Applied computing → Molecular sequence analysis

**Keywords and phrases** Sequence alignment, Substitution matrix, Logistic regression

**Digital Object Identifier** 10.4230/LIPIcs.WABI.2026.17

**Supplementary Material** *Software (Source Code)*: `https://github.com/BioGeMT/DiscrimAlign` [8] archived at `swh:1:dir:e70ee75187e06eb93adc1da6022502a80230d4b0`

**Funding** This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 101244218. We acknowledge the BioGeMT Team (HORIZON-WIDERA-2022 Grant ID: 101086768).

**Acknowledgements** We thank Mr Dave Galea for the help in running simulations and preparing figures.

[1] Corresponding author

© Michał Aleksander Ciach, Elissavet Zacharopoulou, Michał Piotr Startek, Błażej Miasojedow, and Panagiotis Alexiou;
licensed under Creative Commons License CC-BY 4.0

## 1 Introduction

Any kind of alignment of biological sequences requires a substitution matrix and penalties for gap opening and extension. The quality of the resulting alignment and its usefulness for any given task depends greatly on the choice of these parameters [2, 40]. Alignment of DNA sequences can often be performed with a simple three- or four-parameter scoring scheme; for example, the default option in the BLASTN suite (megablast algorithm) is to assign one point to a match, minus two to a mismatch, and minus one and a half for any gap (a so-called linear gap penalty). However, scoring schemes that account for differences in rates of different mutations can improve the performance of database searches [63]. For protein sequences, more complex scoring schemes are needed to account for similarities of amino acids, such as the PAM and BLOSUM series of substitution matrices. In particular, the BLOSUM62 matrix is the *de facto* standard choice [58, 65, 32, 55], at least in part due to being the default option in BLASTP [7, 62], MUSCLE [18], MAFFT [37], MMseqs2 [64] and others. Despite its popularity, it has been reported that, for some protein families, the pairwise alignment with BLOSUM62 aligns less than 50% of residues correctly (evaluated by comparing alignments of sequences and structures), partly because it averages out diverse substitution patterns across families [40, 32, 44]. Furthermore, there is a known bug in the software used to calculate the BLOSUM series, which was found to increase, rather than decrease, its performance in database searches [65]. Accordingly, there is still much to discover and improve about substitution matrices, and numerous alternative substitution matrices have been developed, both general-purpose and application-specific, both for amino acid and nucleotide sequences [69].

There are several approaches to the estimation of substitution matrices and gap penalties, with different optimization criteria and intended applications. Some matrices summarize substitutions observed in a collection of ground-truth alignments, others describe mutations observed in phylogenetic trees, and still others optimize the similarity between sequence alignment and structural alignment of proteins (see the Related work section). Despite the variety of approaches, to our knowledge, there is no approach specifically designed for a binary classification of pairs of sequences. Binary classification covers multiple applications of sequence alignment, including homology search, in which sequences are classified as either related or not; prediction of protein or RNA interactions; read mapping, in which a read can be classified as correctly or incorrectly placed on the genome; detection of duplicated sequences and sequence clustering; motif or domain searches; and others.

**Our contribution.** In this work, we explore the task of discriminative learning of substitution matrices and gap penalties for binary classification of pairs of sequences. We consider a data set $\mathcal{D} = \{(A_i, B_i, y_i) \colon i = 1, 2, \ldots, |\mathcal{D}|\}$, where $A_i$ and $B_i$ are sequences over a common alphabet (typically either DNA, RNA, or amino acid) and $y_i \in \{0, 1\}$ is a binary label associated with each pair. Our goal is to design a supervised learning procedure that can distinguish between positive pairs ($y_i = 1$) and negative pairs ($y_i = 0$) using alignment score. We analyze a logistic link between the sequences and the label, defined as

$$\mathbb{P}(Y_i = 1 |\ \alpha, \theta, A_i, B_i) = \left(1 + e^{-\alpha - S_\theta(A_i, B_i)}\right)^{-1}, \tag{1}$$

where $\theta$ denotes a vector of alignment parameters, $S_\theta(A, B)$ denotes the score of a given alignment model (global, local or glocal) between sequences $A$ and $B$ under parameters $\theta$, and $\alpha$ is an intercept parameter that controls the probability of the pair being positive when the alignment score is zero.

Using a logistic function to classify pairs of sequences after the alignment score has been calculated under a fixed choice of parameters is a well-established idea [16]. In this work, we take a reverse approach: we derive an optimization procedure that, given a dataset of positive and negative examples of unaligned sequences, uses the logistic link to infer the parameters.

Our approach is applicable to local, global and semi-global (glocal) alignment schemes, linear and affine gap penalties, symmetric and asymmetric substitution matrices, and DNA and amino acid alphabets. We present a derivation of the optimization procedure, analyze the properties of the proposed method theoretically and in computational simulation experiments, and present a case study of prediction of microRNA-target interactions, where we show that the proposed method performs comparably to neural networks while retaining the interpretability of sequence alignment. A Python implementation of our method is available at `https://github.com/BioGeMT/DiscrimAlign`, together with Jupyter notebooks with simulation experiments and Python scripts with the case study.

## 2 Related work

The two main approaches to developing amino acid substitution matrices can be described as alignment-based and phylogeny-based. The alignment-based matrices, such as the original BLOSUM [27] and the bug-fixed RBLOSUM and CorBLOSUM series [65, 28, 21], are obtained by counting substitutions in a collection of ground-truth alignments, typically obtained from conserved blocks or by aligning protein structures. This approach has recently been used to obtain the CBM and CCF matrices for *Plasmodium* species [5], asymmetric dirAtPf matrices that capture the compositional bias of *Plasmodium* proteomes [3], the MOLLI series for *Mollicutes* bacteria [46], or the RHOD matrix for microbial rhodopsins [52]. Some alignment-based approaches combine the observed substitutions with other information, for example structural features, which can also account for correlated mutations [68, 32, 39].

Phylogeny-based matrices describe patterns of ancestral substitutions inferred from phylogenetic trees, and can either be in the form of log-odds, like the PAM [15] or VTML [51, 50] series, or in the form of instantaneous rates of mutations, like JTT [35], WAG [70] or LG [45, 12] matrices, the latter being used mostly in maximum likelihood phylogenetic inference. Both forms can be converted to one another by fixing the assumed evolutionary time and taking the matrix exponent of the instantaneous rate matrix to calculate the probabilities of mutations, which can then be transformed to log-odds [42, 50]. The main difference between phylogeny-based and alignment-based matrices is that the former focus on mutations occurring only between an ancestral and a descendent sequence; thus, if an ancestral character $A$ changes to $C$ in one descendant and to $D$ in another, a phylogenetic approach like PAM will consider only the inferred mutations $A \to C$ and $A \to D$, while an alignment-based approach like BLOSUM will consider only the observed exchange $C \leftrightarrow D$ (see e.g. the example discussed in [15]). The phylogenetic approaches focus mostly on general-purpose rather than application-specific matrices, likely due to a considerably more involved estimation procedure [69]. Some approaches combine phylogenetic and alignment-based matrices using principal component analysis [72].

Apart from amino acid matrices, researchers also investigated scoring schemes for nucleotide sequences for homology search [63] and read mapping to reference genomes [20, 25]. Here, ground-truth alignments are often not available, necessitating a different approach. For example, LAST-TRAIN [25] iteratively realigns pairs of sequences and recalculates the matrix, a procedure mathematically based on a Markov Chain formalism of sequence

alignment [16, 19]. A similar approach has been used for disordered proteins [6, 59, 48]. Other applications also require substitution matrices obtained from pairs of sequences rather than multiple sequence alignments or trees, such as predicting protein-protein interactions [31], microRNA-target interactions [67], or DNA-to-protein alignment [73].

The selection of appropriate gap penalties for a given substitution matrix is often considered a separate task. The gap penalties for MOLLI matrices were optimized for ortholog prediction [46]. The POP procedure was developed to optimize the accuracy of alignments on a benchmark of structural alignments [17]. A notable exception is the Inverse Parametric Alignment (IPA) approach, which estimates the substitution matrix jointly with gap penalties by maximizing the score of a given collection of example alignments [40].

While the approaches described above are often used interchangeably, they have different goals and purposes. Phylogeny-based matrices describe the evolutionary process of substitutions, log-odds matrices summarize patterns observed in a collection of alignments, while IPA and POP optimize the similarity of structure-based and sequence-based alignments. While some theoretical guarantees exist for log-odds matrices applied to homology search with an ungapped local alignment [36], overall there is no guarantee that, for example, matrices describing ancestral substitutions, combined with gap penalties optimized for the structural accuracy of an alignment, will perform optimally when applied to the task of homology search with a local alignment. This motivates designing estimation procedures for clearly defined end goals with theoretical guarantees for convergence and optimality.

# 3 Methods

## 3.1 Mathematical properties of pairwise alignment score

In this work, we focus on pairwise sequence alignments that optimize a linear combination of parameters, denoted $\theta$, and counts of associated features such as substitutions or gaps, denoted $x$ and referred to as the *configuration* of the alignment. In the simplest setting, $\theta$ can be a vector of match, mismatch, and linear gap weights, in which case $\theta = (w_{\text{match}}, w_{\text{mismatch}}, w_{\text{gap}})$; the associated configuration vector is then $x = (x_{\text{match}}, x_{\text{mismatch}}, x_{\text{gap}})$, with the numbers of matches, mismatches and gaps in some given alignment. In a setting more commonly used for protein sequences, $\theta$ consists of either 190 or 400 substitution weights (depending on whether the substitution matrix is symmetric) and two gap penalties, and the alignment configuration $x$ counts the associated substitutions, gap openings, and gap extensions.

Let $\Theta$ denote the space of possible parameters and let $\mathcal{X}(A, B)$ denote the set of all alignment configurations that can be obtained from $A$ and $B$. As the scoring scheme influences the shape of the parameter space $\Theta$, the alignment scheme (local, global or glocal) influences the space $\mathcal{X}(A, B)$. For example, the zero vector corresponding to an empty alignment always belongs to the space of configurations of a local alignment, but not of the global one (and, in general,$\mathcal{X}_{\text{global}}(A, B)$ is a proper subset of $\mathcal{X}_{\text{local}}(A, B)$).

The specification of $\Theta$ and $\mathcal{X}(A, B)$ is equivalent to the specification of alignment and scoring schemes. In this work, we will assume that $\Theta = \mathbb{R}^n$ for some $n$. Let $\langle x, y\rangle = \sum_i x_i y_i$ denote the inner product. Given a fixed configuration space $\mathcal{X}(A, B)$ and some $\theta \in \Theta$, the alignment score of sequences $A, B$ can be defined as

$$S_\theta(A, B) = \langle \theta, x^* \rangle = \max_{x \in \mathcal{X}(A,B)} \langle \theta, x \rangle. \tag{2}$$

This kind of formulation has previously been used in research on Parametric and Inverse-Parametric Alignment [38, 40, 33]. In general, the space $\mathcal{X}(A, B)$ has a complex structure and its size grows exponentially with the length of the sequences [23]. The classical alignment

algorithm based on dynamic programming serves as a so-called *oracle* that can return an optimal configuration in polynomial time [40]. We remark that different alignments can have the same configuration, and different configurations can potentially yield the same optimal score, so $x^*$ in the definition above denotes one of the optimal configurations that may itself correspond to several alignments.

The formulation of pairwise alignment with Equation (2) allows us to state the following Proposition, which describes the shape of regions of optimality for a given configuration. A set $U$ is *convex* if $x, y \in U$ implies $px + (1-p)y \in U$ for all $p \in [0,1]$; a set $U$ is a *cone* if $x \in U$ implies $sx \in U$ for all $s > 0$.

▶ **Proposition 1.** *Let $U_{A,B}(x) \subseteq \Theta$ denote the subset of the parameter space on which the alignment configuration $x$ is optimal, and let $U^*_{A,B}(x) \subseteq \Theta$ denote the subset in which $x$ is uniquely optimal. Then, $U_{A,B}(x)$ is a non-empty closed convex cone, and $U^*_{A,B}(x)$ is an open convex cone.*

This result was shown previously in the context of parametric alignment [54] and generalizes previous results about the decomposition of the parameter space [66, 24]. The proof is based on comparing $x$ to every other configuration and expressing $U_{A,B}(x)$ as an intersection of the resulting hyperplanes (see the Appendix for details). The cone property is a mathematical expression of the fact that scaling all parameters by a positive constant does not change the relative scores of different alignments, thus preserving the optimal one [2]. The next Proposition describes the dependence of $S_\theta(A,B)$ on $\theta$, and will later be necessary to derive generalized gradients for our optimization procedure. Let $||\cdot||$ denote the Euclidean (L2) norm. A function $f\colon \mathbb{R}^n \to \mathbb{R}$ is said to be Lipschitz continuous with a constant $L$ if for any $\theta_1, \theta_2 \in \mathbb{R}^n$ we have $|f(\theta_1) - f(\theta_2)| \leq L||\theta_1 - \theta_2||$ (i.e. the growth of $f$ is linearly bounded by $L$). A function $f$ is said to be convex if, for any $\theta_1, \theta_2 \in \mathbb{R}^n$ and $p \in [0,1]$, we have $f(p\theta_1 + (1-p)\theta_2) \leq pf(\theta_1) + (1-p)f(\theta_2)$.

▶ **Proposition 2.** *The function $\theta \to S_\theta(A,B)$ is convex and Lipschitz continuous with the Lipschitz constant* $\max_{x \in \mathcal{X}(A,B)} ||x||$.

Both properties stem from the fact that for any vector $x$, the scalar product $\langle \theta, x \rangle$ is a convex and Lipschitz continuous function of $\theta$, and the maximum operator over a finite number of functions preserves these properties (see the Appendix for a detailed proof). The Rademacher theorem states that Lipschitz-continous functions are differentiable almost everywhere (i.e. the set of points of non-differentiability has a Lebesgue measure zero). In the case of $S_\theta(A,B)$, if $\theta \in U^*(x)$ for some $x$, then $\nabla_\theta S_\theta(A,B)$ exists at $\theta$ and is equal to $x$. On the other hand, if two or more different configurations are optimal at $\theta$, then $S_\theta(A,B)$ is non-differentiable at $\theta$. It follows that the optimal configurations are unique for almost all values of $\theta$.

The non-existence of gradients of $S_{A,B}(\theta)$ complicates the use of gradient descent methods for parameter learning. This problem has recently been overcome by designing differentiable alignment algorithms [57, 53, 41, 49]. In this work, we will derive a learning procedure for the standard alignment algorithms using *subdifferentials*, a generalization of gradients for non-differentiable convex functions.

▶ **Definition 3** (Adapted from [4]). *A subdifferential of a convex function $f\colon \mathbb{R}^n \to \mathbb{R}$ at a point $\theta \in \mathbb{R}^n$, denoted $\partial f(\theta)$, is the set of coefficient vectors of linear functions which bound $f$ from below (also referred to as supporting hyperplanes of $f$ at $\theta$):*

$$\partial f(\theta) = \{v \in \mathbb{R}^n \colon \forall_{\theta' \in \mathbb{R}^n} \; f(\theta') \geq f(\theta) + \langle v, \theta' - \theta \rangle\}. \tag{3}$$

*A subgradient of $f$ at $\theta$ is any element of the subdifferential $\partial f(\theta)$.*

For example, if $n = 1$ and $f(\theta) = |\theta|$, then $\partial f(\theta) = \{-1\}$ for all $\theta < 0$, $\partial f(\theta) = \{1\}$ for all $\theta > 0$, and $\partial f(0) = [-1, 1]$. Similarly to standard gradients, subgradients indicate the direction of growth of the function: for any $v \in \partial f(\theta)$, setting $\theta' = \theta + \lambda v$ in Equation (3) results in $f(\theta + \lambda v) - f(\theta) \geq \langle v, \lambda v \rangle = \lambda ||v||^2$, indicating that $f(\theta + \lambda v) \geq f(\theta)$ whenever $\lambda \geq 0$. If follows that $\theta$ is the minimum of $f(\theta)$ if and only if $0 \in \partial f(\theta)$ [4]. However, unlike standard gradients of differentiable functions, the definition does not guarantee that $-v$ indicates the direction in which the function decreases.

▶ **Proposition 4.** *Let $S_\theta(A, B)$ be the optimal score of alignment of sequences $A$ and $B$ with parameter vector $\theta$. Let $\mathrm{Conv}(X)$ denote the convex hull of a set $X$, i.e. the minimal convex set containing all points from $X$ (equivalently, the set of all convex combinations of points from $X$). Then, the subdifferential of the function $\theta \to S_{A,B}(\theta)$ is the convex hull of all configurations optimal at $\theta$:*

$$\partial S_\theta(A, B) = \mathrm{Conv}(\{x^* \in \mathcal{X}(A, B) : \langle \theta, x^* \rangle = S_\theta(A, B)\}).$$

The Proposition follows from the fact that for a function $f$ defined as $f(\theta) = \max_{i \in I} f_i(\theta)$, if $I$ is a finite set and $f_i$ are convex, then the subdifferential of $f$ is the convex hull of subdifferentials of functions $f_i$ active at $\theta$, i.e. $\partial f(\theta) = \mathrm{Conv} \bigcup_i \{\partial f_i(\theta) \colon \ f_i(\theta) = f(\theta)\}$ (see e.g. Theorem 3.50 in ref. [4]). From the Proposition it follows that any optimal configuration is a subgradient of $S_\theta(A, B)$, and can be obtained by calculating the score with the dynamic programming algorithm, backtracking the optimal alignments, and calculating the configurations. If only one subgradient is necessary for any given $\theta$, it can be calculated without backtracking by keeping track of the choices during the calculation of the score with the dynamic programming algorithm.

## 3.2 The likelihood function of the proposed model

Let $p(\alpha, \theta;\ A_i, B_i) = \mathbb{P}(Y_i = 1|\ \theta, \alpha, A_i, B_i)$. In the same way as in logistic regression models, we assume that the likelihood of parameters $\alpha$ and $\theta$ given a dataset $\mathcal{D}$ is equal to the product of probabilities defined by Equation (1):

$$\mathcal{L}(\alpha, \theta; \mathcal{D}) = \prod_{i=1}^{|\mathcal{D}|} \mathbb{P}(Y_i = y_i|\ \alpha, \theta, A_i, B_i) \tag{4}$$

$$= \prod_{i=1}^{|\mathcal{D}|} p(\alpha, \theta;\ A_i, B_i)^{y_i} (1 - p(\alpha, \theta;\ A_i, B_i))^{1 - y_i} \tag{5}$$

Given a dataset $\mathcal{D}$, the parameters $\alpha$ and $\theta$ can be estimated by maximizing the likelihood, equivalent to maximizing the log-likelihood $l(\alpha, \theta; \mathcal{D}) = \ln \mathcal{L}(\alpha, \theta; \mathcal{D})$. Let $\mathcal{X}_i = \mathcal{X}(A_i, B_i)$. After taking the logarithm of Equation (5), substituting Equations (1) and (2), and doing some algebraic manipulations, the log-likelihood of the proposed model can be expressed as

$$l(\alpha, \theta; \mathcal{D}) = \sum_{i=1}^{|\mathcal{D}|} \Big( y_i \ln p(\alpha, \theta;\ A_i, B_i) + (1 - y_i) \ln(1 - p(\alpha, \theta;\ A_i, B_i)) \Big)$$

$$= \sum_{i=1}^{|\mathcal{D}|} y_i \Big( \alpha + \max_{x \in \mathcal{X}_i} \langle \theta, x \rangle \Big) - \sum_{i=1}^{|\mathcal{D}|} \ln \Big( 1 + \exp(\alpha + \max_{x \in \mathcal{X}_i} \langle \theta, x \rangle) \Big)$$

The main difference between the proposed model and the standard logistic regression is the presence of the maximum operator within the log-likelihood. In fact, fixing the alignments

reduces the model to a standard logistic regression. Namely, given a list of fixed alignments $\mathcal{A}_i = \mathcal{A}(A_i, B_i)$, the probability of a pair being positive can be expressed as

$$\mathbb{P}(Y_i = 1|\ \alpha, \theta, \mathcal{A}_i) = \left(1 + e^{-\alpha - \langle \theta, x(\mathcal{A}_i)\rangle}\right)^{-1} \tag{6}$$

This equation corresponds to a standard logistic regression model with intercept $\alpha$, vector of parameters $\theta$, and independent variables $x(\mathcal{A}_i)$. If a list of ground-truth alignments is available, this model can be used to estimate the parameters, but does not guarantee that the alignments will be optimal for parameters estimated this way. Conversely, the model in Equation (5) uses unaligned pairs of sequences and estimates the alignments jointly with the parameters, allowing both to change during optimization.

Note that functions $y_i(\alpha + S_\theta(A_i, B_i))$ are convex as affine transformations of a convex function $S_\theta(A_i, B_i))$, functions $\ln(1 + \exp(\alpha + S_\theta(A_i, B_i)))$ are convex as compositions of a strictly increasing convex function $z \to \ln(1 + \exp(\alpha + z))$ and convex functions $S_\theta(A_i, B_i)$, and convexity is preserved under summation. The log-likelihood of the proposed model is therefore a difference of two convex functions, which may be neither convex nor concave (as opposed to the concave log-likelihood of the standard logistic regression). However, it is concave locally almost everywhere, i.e. for almost all $\theta \in \mathbb{R}^n$ (in the sense of Lebesgue measure) there exists an open neighborhood $U \ni \theta$ such that the function restricted to $U$ is concave (i.e. $-l(\alpha, \theta; \mathcal{D})_{|\theta \in U}$ is convex).

▶ **Proposition 5.** *The function $l(\alpha, \theta; \mathcal{D})$ is smooth and locally concave with respect to $\theta$ almost everywhere.*

The Proposition stems from the fact that the log-likelihood is differentiable almost everywhere, and in the points of differentiability it locally reduces to the log-likelihood of a standard logistic regression. A detailed proof is presented in the Appendix. Since $l(\alpha, \theta; \mathcal{D})$ is not globally concave, the model may not be uniquely identifiable (i.e. there may be more than one maximizer of the log-likelihood). An intuitive explanation of this phenomenon is that the added flexibility of alignment means that different substitution matrices may perform equally well in classifying pairs of sequences, as the differences in substitution matrices can potentially be offset by adjusting the alignments. As a consequence, we will look for example maximizers $\alpha^*, \theta^* \in \arg\max_{\alpha,\theta} \mathcal{L}(\alpha, \theta; \mathcal{D})$. An exact characterization of the geometry of the set of maximizers remains an open problem. The following Proposition states that this set is small in the sense of Lebesgue measure:

▶ **Proposition 6.** *The set of maximizers $\arg\max_{\alpha,\theta} \mathcal{L}(\alpha, \theta; \mathcal{D})$ has Lebesgue measure zero.*

The Proposition follows from the fact that $\Theta$ is composed of a finite number of open subsets on which $l(\alpha, \theta; \mathcal{D})$ is concave (and thus has a unique maximizer), and the boundaries of these subsets, which have measure zero. The next Proposition states that a necessary condition for the boundedness of the set of maximizers is the lack of a complete separation of the labels, similarly as in the standard logistic regression [1]:

▶ **Proposition 7.** *Assume there exists a threshold $c \in \mathbb{R}$ and a parameter vector $\theta$ such that $S_\theta(A_i, B_i) > c$ for any $i$ such that $y_i = 1$ and $S_\theta(A_i, B_i) < c$ for any $i$ such that $y_i = 0$. Then, $\sup_{\alpha,\theta} \mathcal{L}(\alpha, \theta; \mathcal{D}) = 1$. Furthermore, for any sequence $\alpha_k, \theta_k$ such that $\lim_{k\to\infty} \mathcal{L}(\alpha_k, \theta_k; \mathcal{D}) = 1$, the alignment parameters diverge to infinity: $||\theta_k|| \to \infty$.*

**Proof.** Let $\alpha_k = -kc$, $\theta_k = k\theta$ and observe that $S_{\theta_k}(A_i, B_i) = kS_\theta(A_i, B_i)$. We have

$$\mathbb{P}(Y_i = 1|A_i, B_i, \alpha_k, \theta_k) = (1 + \exp(k(c - S_\theta(A_i, B_i))))^{-1},$$
$$\mathbb{P}(Y_i = 0|A_i, B_i, \alpha_k, \theta_k) = (1 + \exp(-k(c - S_\theta(A_i, B_i))))^{-1}.$$

Since $c - S_\theta(A_i, B_i) < 0$ for $y_i = 1$ and $c - S_\theta(A_i, B_i) > 0$ for $y_i = 0$, for every $i$ we have $\mathbb{P}(Y_i = y_i | A_i, B_i, \alpha_k, \theta_k) \to 1$ as $k \to \infty$, which proves the first part of the Proposition. The second part follows from the fact that the likelihood function attains 1 only if all the logistic probabilities attain 1, which can only happen in the limit of infinite alignment scores. ◀

We also conjecture, motivated by similar results in standard logistic regression [1] and the results of our numerical experiments, that a so-called overlap of points (i.e. a dataset in which any set of parameters results in at least one misclassified label) is sufficient for there to exist finite maximizers $\alpha^*, \theta^*$ (and, possibly, for the set of maximizers to be bounded). In what follows, we will assume that the set of maximizers of $l(\alpha, \theta; \mathcal{D})$ is non-empty.

## 3.3 The Clarke subdifferential of the log-likelihood

As opposed to the function $\theta \to S_\theta(A, B)$, the functions $\ln p(\theta)$ and $\ln(1 - p(\theta))$ may not be convex nor concave. This is because the logarithm of the logistic function is concave, while the alignment score is convex, and a composition of a concave and a convex function may be neither. Thus, these functions may not have standard subdifferentials, and in order to analyze them we need to use a further generalization called the Clarke subdifferentials [9, 10]. A function is said to be locally Lipschitz continuous if, for every $\theta \in \mathbb{R}^n$, there exists an open neighbourhood $U \ni \theta$ such that $f$ is Lipschitz continuous on $U$.

▶ **Definition 8** (Adapted from [9]). *For a locally Lipschitz function $f\colon \mathbb{R}^n \to \mathbb{R}$, the Clarke subdifferential at a point $\theta$, denoted $\partial^C f(\theta)$, is defined as:*

$$\partial^C f(\theta) = \mathrm{Conv}\{v \in \mathbb{R}^n \colon \exists_{\theta_n \to \theta}\ \nabla f(\theta_n) \to v\}. \tag{7}$$

Intuitively, one may think about the Clarke subdifferential as the set of all gradients of $f$, and their convex combinations, around $\theta$. This intuition is substantiated by the fact that locally Lipschitz functions are differentiable almost everywhere, so their gradients exist almost everywhere. Points $\theta$ such that $0 \in \partial^C f(\theta)$ are called *Clarke-stationary* or *Clarke-critical*. If $f$ has a local minimum or maximum at $\theta$, then $\theta$ is a Clarke-stationary point, but the converse may not hold (Proposition 2.3.2 in [10]). As opposed to standard calculus, the rules of Clarke calculus often provide only inclusions rather than equalities.

▶ **Lemma 9** (Consequence of Theorem 2.3.9 in [10]). *Let $f, g\colon \mathbb{R}^n \to \mathbb{R}$ be locally Lipschitz, and let $h\colon \mathbb{R} \to \mathbb{R}$ be locally Lipschitz and continuously differentiable. Let $\oplus$ denote the Minkovsky sum of sets defined as $A \oplus B = \{a + b \colon a \in A, b \in B\}$. Then,*

$$\partial^C(f(\theta) + g(\theta)) \subseteq \partial^C f(\theta) \oplus \partial^C g(\theta) \tag{8}$$

$$\partial^C h(f(\theta)) = h'(f(\theta))\partial^C f(\theta) \tag{9}$$

We remark that the chain rule above works only in the special case that $h$ is a continuously differentiable function of a single variable; in general, only an inclusive chain rule is satisfied with $\partial^C h(f(x)) \subseteq h'(f(x))\partial^C f(x)$. In what follows, we will denote the Clarke subdifferential with respect to $\theta$ as $\partial^C_\theta$, treating the parameter $\alpha$ as a constant.

▶ **Proposition 10.** *The Clarke subdifferentials of $\ln p(\alpha, \theta; A, B)$ and $\ln(1 - p(\alpha, \theta; A, B))$ with respect to $\theta$ are equal to*

$$\partial^C_\theta \ln p(\alpha, \theta; A, B) = (1 - p(\alpha, \theta; A, B))\partial S_\theta(A, B),$$
$$\partial^C_\theta \ln(1 - p(\alpha, \theta; A, B)) = -p(\alpha, \theta; A, B)\partial S_\theta(A, B).$$

The proof is based on showing that the functions are locally Lipschitz continuous and applying Lemma 9. Details are presented in the Appendix. Using a variable $y \in \{0, 1\}$, we can combine both equations from the Proposition into

$$\partial_\theta^C \big(y \ln p(\alpha, \theta; A, B) + (1 - y) \ln(1 - p(\alpha, \theta; A, B))\big) = (y - p(\alpha, \theta; A, B))\partial S_\theta(A, B).$$

This equality follows from the fact that any choice of $y \in \{0, 1\}$ reduces this equation to either of the two from the Proposition. From Lemma 9 we now have

$$\partial_\theta^C \; l(\theta; \mathcal{D}) \subseteq \bigoplus_{(A_i, B_i, y_i) \in \mathcal{D}} (y_i - p(\alpha, \theta; A_i, B_i))\, \partial S_\theta(A_i, B_i).$$

In general, $\partial_\theta^C \; l(\theta; \mathcal{D})$ is a proper subset of the Minkovsky sum on the right-hand side. This stems from the fact that the Minkovsky sum allows us to select any optimal configuration for each sequence pair, while the Clarke subdifferential requires a single convergent sequence of parameters $\theta_n \to \theta$ for which the configurations are jointly uniquely optimal:

▶ **Lemma 11.** *Let* $\mathcal{B}(\theta; \mathcal{D}) = \{(x_1, x_2, \ldots, x_{|\mathcal{D}|}) \colon x_i \in \partial_\theta S(\theta; A_i, B_i)\}$. *Then,*

$$\exists_{\theta_n \to \theta} \forall_n \mathcal{B}(\theta_n; \mathcal{D}) = \{(x_1, \ldots, x_{|\mathcal{D}|})\} \;\Rightarrow\; \sum_{i=1}^{|\mathcal{D}|} (y_i - p(\alpha, \theta; A_i, B_i)) x_i \in \partial_\theta^C l(\alpha, \theta; \mathcal{D})$$

A proof of the Lemma is presented in the Appendix. The Lemma can be used to show the following Proposition. We remark that both the Lemma and the Proposition state an inclusion rather than equality due to the technical fact that the sum of non-differentiable functions may be differentiable. This issue is discussed further in the proof of the Lemma.

▶ **Proposition 12.** *Let* $(x_1, x_2, \ldots, x_{|\mathcal{D}|}) \in \mathcal{B}(\theta; \mathcal{D})$. *For* $x_i \in \mathcal{X}_i$, *let* $U_i^*(x_i) \subseteq \Theta$ *be the set of parameters* $\theta$ *for which* $x_i$ *is a uniquely optimal configuration for* $A_i, B_i$. *Then,*

$$\bigcap_{i=1}^{|\mathcal{D}|} U_i^*(x_i) \neq \emptyset \;\Rightarrow\; \sum_{i=1}^{|\mathcal{D}|} (y_i - p(\alpha, \theta; A_i, B_i)) x_i \in \partial_\theta^C l(\alpha, \theta; \mathcal{D})$$

We note that the above subgradients of $l(\alpha, \theta; \mathcal{D})$ have an intuitive interpretation. The weights $y_i - p(\alpha, \theta; A_i, B_i)$ are the errors of the model, which are positive for the positive pairs and negative for the negative ones. Thus, they prioritize increasing the parameters corresponding to common substitutions in low-scoring positive pairs and decreasing the ones corresponding to common substitutions in high-scoring negative pairs. We also note a connection to the problem of separability: if there exists a sequence $\theta_k$ such that $\lim_{k\to\infty} p(\alpha, \theta_k; A_i, B_i) = y_i$ for all $i$, then $\lim_{k\to\infty} \partial_\theta^C l(\alpha, \theta_k; \mathcal{D}) = 0$, resulting in "Clarke-stationarity in infinity".

We remark that functions whose gradients point towards the direction of increase are termed *pseudoconvex*. The following Proposition establishes that $\ln p(\alpha, \theta; A, B)$ is a Clarke-pseudoconvex function and $\ln(1 - p(\alpha, \theta; A, B))$ is a Clarke-pseudoconcave function. Whether $\partial_\theta^C l(\alpha, \theta; \mathcal{D})$ admits a similar property remains an open question.

▶ **Proposition 13.** *Let* $d_{\ln p} \in \partial_\theta^C \ln p(\alpha, \theta; A, B)$ *and* $d_{\ln(1-p)} \in \partial_\theta^C \ln(1 - p(\alpha, \theta; A, B))$. *Then, for any* $\theta' \in \Theta$*:*

- *If* $\langle d_{\ln p}, \theta' - \theta \rangle \geq 0$, *then* $\ln p(\alpha, \theta'; A, B) \geq \ln p(\alpha, \theta; A, B)$,
- *If* $\langle d_{\ln(1-p)}, \theta' - \theta \rangle \leq 0$, *then* $\ln(1 - p(\alpha, \theta'; A, B)) \leq \ln(1 - p(\alpha, \theta; A, B))$.

The proof is based on an observation that $d_{\ln p}$ is proportional to a subgradient of the alignment score, and thus points in the direction of increasing alignment score, which in turn is the direction of increase of $\ln p(\alpha, \theta; A, B)$ for a fixed $\alpha$.

## 3.4 Fitting the model with the subgradient method

While there are specialized methods of optimizing differences of convex functions such as the log-likelihood of the proposed model [29, 34], in this work we present a general subgradient method because it results in a particularly simple algorithm that can be implemented in a few lines of computer code. The subgradient method is a generalization of gradient ascent that essentially replaces the gradient with any subgradient. While the definition of subgradients does not guarantee that the function value increases in each step (hence the name "subgradient method" rather than "subgradient ascent"), the method converges to a Clarke stationary point under some general assumptions. We will first present the conditions for finding optimal $\theta$ assuming that the $\alpha$ parameter is known. Below, "almost surely" means "with probability 1" and refers to the randomness of the trajectory, and "limit points" of $\theta_k$ refer to limits of subsequences of $\theta_k$ (which may exist even if $\theta_k$ does not converge to a single point).

▶ **Theorem 14.** *For $k \geq 1$, let $\xi_k$ be an $n$-element vector of independent random variables with zero expected value and standard deviation $\sigma_k$, where $n$ is the dimension of $\Theta$. Let $\theta_0 \in \Theta$ and let the sequence $\theta_k$ be defined as*

$$\theta_{k+1} = \theta_k + \eta_k g_k + \xi_k \text{ for } g_k \in \partial^C_\theta l(\alpha, \theta_k; \mathcal{D}) \tag{10}$$

*Assume that $\eta_k > 0, \sum \eta_k = \infty, \sum \eta_k^2 < \infty, \sigma_k/\eta_k \to 0$, and that the sequence $\theta_k$ is bounded. Then, almost surely, every limit point of $\theta_k$ is a Clarke-stationary point of $l(\alpha, \theta; \mathcal{D})$.*

The proof is based on showing that the sequence of parameters satisfies Assumptions C and D from ref. [14], and presented in the Appendix. An example sequence of step-lengths that satisfies the assumptions of the Theorem is $\eta_k = \eta/k$ for some $\eta > 0$ and $\sigma_k = \sigma/k^2$ for some $\sigma \geq 0$. Note that while $\sigma_k \equiv 0$ also satisfies the assumptions of the Theorem, some amount of random noise may be beneficial in practice e.g. to avoid local stationary points. The assumption $\sigma_k/\eta_k \to 0$ can be considerably weakened to still preserve convergence, but having the random noise vanish faster than the step size allows the method to refine the estimates without random perturbations in final iterations. Finally, while subgradients of the log-likelihood can be theoretically constructed using Proposition 12, it might be computationally expensive in practice. However, if $\xi_k$ are continuous random variables, then for $k \geq 1$ the log-likelihood is differentiable at $\theta_k$ almost surely. This reduces the condition $g_k \in \partial^C_\theta l(\alpha, \theta_k; \mathcal{D})$ to $g_k = \nabla_\theta l(\alpha, \theta_k; \mathcal{D}) = \sum_i (y_i - p(\alpha, \theta_k; A_i, B_i)) x_i$ almost surely.

We remark that the update step above uses a standard gradient of the log-likelihood. The algorithm can be derived heuristically by simply differentiating the log-likelihood where possible and using the gradient in a heuristic gradient ascent method. However, gradient ascent methods can oscillate or diverge for some "pathological" non-smooth non-convex functions [13, 60]. The formalism presented in this paper establishes convergence guarantees of the proposed method, which are, however, more subtle than for smooth functions. For example, Theorem 14 guarantees that limits of subsequences of $\theta_k$ are Clarke-stationary, and a similar proof can be made to show that the whole sequence $\theta_k$ accumulates around the set of Clarke-stationary points, but, in general, does not guarantee that the whole sequence $\theta_k$ converges to a single point (see e.g. the example in ref. [60] in which the trajectory oscillates around a circle of stationary points). The existence of stronger convergence guarantees for the proposed model, and sufficient conditions for the existence of stationary points and boundedness of $\theta_k$, remains an open question. In particular, the lack of complete separability is a necessary condition for the existence of finite global maximizers and boundedness of $\theta_k$, but does not preclude the existence of local maxima or other Clarke-stationary points.

Theorem 14 requires that all alignments be recalculated at each step, making it computationally intensive for large datasets. This can be mitigated by taking a random sample of sequence pairs at each iteration, similar to a batch learning strategy in neural networks:

▶ **Theorem 15.** *For $k \geq 1$ and $N \in \mathbb{N}_+$, let $\mathcal{D}_k$ be a sequence of independent $N$-element random subsets of $\mathcal{D}$ such that the elements of each subset are selected uniformly with replacement. Assume the same conditions on $\eta_k$ and $\xi_k$ as in Theorem 14, but let the sequence $\theta_k$ now be defined as*

$$\theta_{k+1} = \theta_k + \eta_k g_k + \xi_k \text{ for } g_k \in \partial_\theta^C l(\alpha, \theta_k; \mathcal{D}_k) \tag{11}$$

*Assume that subgradients $g_k$ are obtained using Proposition (12) applied to $l(\alpha, \theta_k; \mathcal{D}_k)$. Then, with probability 1, every limit point of $\theta_k$ is a Clarke-stationary point of $l(\alpha, \theta; \mathcal{D})$.*

The proof is presented in the Appendix. Theorems 14 and 15 provide ways of estimating $\theta$ under a known $\alpha$. In order to optimize the latter, we note that the log-likelihood is differentiable and concave (due to a negative second derivative) with respect to $\alpha$ for any fixed $\theta$. It follows that for any fixed $\theta$ the optimal value of $\alpha$ is unique and can be efficiently found with standard optimization methods such as the Newton-Rhapson procedure. We use this observation to replace the free parameter $\alpha$ with $\alpha^*(\theta) = \arg\max_\alpha l(\alpha, \theta; \mathcal{D})$ and optimize $l(\alpha^*(\theta), \theta; \mathcal{D})$ using the subgradient method for $\theta$. This procedure is summarized in Algorithm 1.

## 4 Results

While theoretical guarantees of convergence require a decreasing sequence of steps, in practice we generally see a better performance with a constant step size $\eta_k \equiv \eta$ due to a faster convergence, and we use this sequence in the Results section with $\sigma_k = \sigma/k$.

**Algorithm 1** Discriminative learning of alignment parameters from pairs of labeled sequences.

**Input:** List of sequence pairs with binary labels $\mathcal{D} = \{(A_i, B_i, y_i) \colon i = 1, 2, \ldots, |\mathcal{D}|\}$;
**Hyperparameters:** Maximum number of iterations $I_{\max}$; Sequence of step lengths $\eta_k$; Sequence of standard deviations $\sigma_k$; Initial alignment parameters
**Output:** Alignment parameters $\theta^*$, logistic intercept $\alpha^*$;
**Let** $\mathcal{A}_0 \leftarrow$ alignments of $(A_i, B_i)$ under initial parameters
**Let** $\alpha_0, \theta_0 \leftarrow$ Initial estimate using Equation (6)
**for** $k = 1, 2, \ldots, I_{\max}$ **do**
  **Let** $\mathcal{A}_k \leftarrow$ Alignments of $\mathcal{D}$ under $\theta_{k-1}$
  **Let** $x^* \leftarrow$ configurations of $\mathcal{A}_k$
  **Let** $g_k \leftarrow \sum_i (y_i - p(\theta_{k-1}; \alpha_{k-1}))x_i^*$
  **Let** $\xi_k \sim \mathcal{N}(0, \sigma_k I)$
  **Let** $\theta_k \leftarrow \theta_{k-1} + \eta_k g_k + \xi_k$
  **Let** $\alpha_k \leftarrow \max_\alpha l(\theta_k, \alpha)$ ▷ Newton-Rhapson procedure
**end for**
**Return:** $\theta^* = \theta_k, \alpha^* = \alpha_k$

### 4.1 Convergence and accuracy of estimation

To experimentally validate the proposed method, we performed simulation experiments with fixed substitution matrices and gap penalties using a data set of microRNA (miRNA) sequences. MiRNAs are short RNA molecules (typically 18-25 nt) that regulate gene

expression by recognizing mRNA transcripts, or other RNA molecules, through partial complementary pairing and inhibiting translation and/or promoting degradation [47, 43, 30]. Recently, a set of benchmark datasets called miRBench was released, which contains several datasets of pairs of miRNA sequences and 50nt-long RNA sequences labeled based on whether the RNA sequence contains a target site for the miRNA [61, 26]. We used the train split of the Hejret dataset from miRBench, which contains 4084 positive and 4109 negative instances.

In order to obtain a dataset with a known ground truth, we replaced the original binary labels with simulated ones generated as follows. We investigated two local alignment scoring schemes, referred to as "simple model" and "full model". The "simple" model had match weight 1, mismatch weight -1, gap open -1.2 and gap extend -0.1 The more challenging "full" model had an asymmetric DNA substitution matrix $M$ with manually selected entries to obtain a diversity of scores:

$$M = \left(\begin{array}{c|cccc} & A & C & U & G \\ \hline A & 1.0 & -0.3 & -1.0 & -0.8 \\ C & -0.6 & 1.2 & -0.3 & -1.0 \\ U & -1.2 & -0.4 & 1.0 & -0.8 \\ G & -0.4 & -1.4 & -0.9 & 1.3 \end{array}\right) \qquad G = \left(\begin{array}{r|l} \text{Gap open} & -1.2 \\ \text{Gap extend} & -0.1 \end{array}\right)$$

To generate labels, we calculated the scores of pairwise local alignments of miRNA sequences and RNA fragments using Biopython [11]. We used the scores to calculate the probabilities of interactions using Equation (1), with $\alpha$ equal to the negative of the median score to obtain a roughly balanced dataset. Finally, for each pair, we used the corresponding probability to simulate a binary label. We run DiscrimAlign on the simulated dataset for 200 iterations with a stochastic factor $\sigma = 0.001$ and different learning rates $\eta$ ranging from 5e-06 to 5e-05 (using a constant step size in order to obtain a faster convergence, as discussed in Methods).

The results are shown in Figure 1. Step sizes which resulted in an initial drop in the log-likelihood often corresponded to a faster convergence. Excessive step sizes resulted in an unstable behavior, while insufficient step sizes result in a very slow convergence. The step sizes $\eta = 2 \cdot 10^{-5}$ for the simple model and $\eta = 4 \cdot 10^{-5}$ for the full model were among the smallest step sizes that resulted in a sufficiently fast convergence, thus offering a balance between speed and numerical stability. Fig. 2 shows the estimated weights of the asymmetric matrix for $\eta = 4 \cdot 10^{-5}$. The correlation between the entries of the true substitution matrix $M$ and the estimated one $\hat{M}$ was equal to over 0.99, and the mean absolute error of estimation was 0.06. To analyze the robustness of the method, we run DiscrimAlign with $\eta = 4 \cdot 10^{-5}$ for 20 replicated datasets with the asymmetric matrix. The results show that the speed of convergence varies, but the true value of the log-likelihood was reached (or nearly reached) after 200 iterations in all replicates (Fig. 1). The mean absolute error varied between 0.06 and 0.12.

## 4.2 Dataset requirements for amino acid matrices

A natural question is the size of dataset necessary for an accurate estimation depending on the size of the model and the alphabets. For example, a full model for amino acid sequences would require 403 parameters (400 substitution weights, 2 gap penalties, and the intercept $\alpha$). The error of estimation can be decomposed into error resulting from the training procedure (which includes learning the alignments) and the inherent statistical error resulting from the finite size of the dataset and randomness of labels. In this section, we show how to estimate the latter error in simulated experiments using a standard logistic regression model applied to ground-truth alignments (Equation (6)). This provides a lower bound on the error of

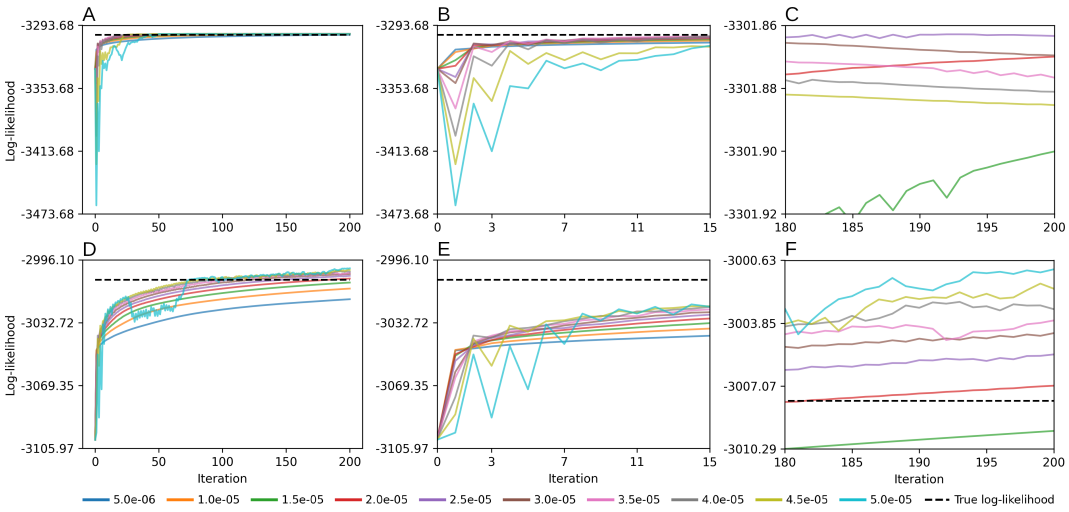



**Figure 1 Estimation of discriminative substitution matrices with DiscrimAlign for pairs of RNA sequences with simulated labels.** Plots show trajectories of the log-likelihood for different step sizes $\eta$ for a constant sequence of step sizes. A,B,C: Simple substitution model with match/mismatch weights and an affine gap penalty (4 parameters plus intercept). D,E,F: Asymmetric substitution matrix and an affine gap penalty (18 parameters plus intercept). A,D: Full trajectories; B,E: Initial iterations; C,F: Final iterations. All trajectories shown in panel C are above the true log-likelihood; the remaining trajectories did not reach it or were unstable. The light blue line in D-F shows the method's behavior for excessive step sizes (numerical instability), while the dark blue line shows insufficient step sizes (slow convergence).

DiscrimAlign without the need for a full training procedure, while also allowing one to use an extensive body of statistical results including confidence intervals. In turn, a lower bound on the error provides a lower bound on dataset requirements.

To generate simulated ground-truth labels to study the lower bound on estimation error for protein alignments, we used a centered and scaled BLOSUM62 matrix with gap penalties $G = (-1, -0.1)$. Scaling the matrix was done so that individual substitutions do not have an overwhelming impact on the predicted probability, as large values of $\theta$ mean that any differences in alignment configurations result in large changes of the predicted probability. To generate sequence pairs, we downloaded proteomes of *Homo sapiens* (`GCF_000001405.40`, 136 807 sequences) and *Gallus gallus* (`GCF_016699485.2`, 68 683 sequences) from the NCBI Genome database. We selected proteins up to 1 000 amino acids in length to avoid excessive computational times, used a local installation of BLASTP to identify homologs with an E-value threshold of 1e-6 (resulting in 2 238 760 pairs of proteins), and created three datasets by randomly selecting pairs with $|\mathcal{D}| = 10^4, 10^5$ or $10^6$. We remark that these sequence pairs have highly varying degrees of similarity, because an E-value threshold of 1e-6 is relatively high when comparing two proteomes rather than a proteome against a large database (since E-values are proportional to the number of compared sequences). We calculated global alignments and discarded those which consisted of more than 20% of gaps. We used the scores of remaining alignments to generate labels as in the previous section. Finally, we arranged the configurations of the alignments and the labels into arrays and estimated the parameters using the `LogisticRegression` function implemented in the scikit-learn library (`penalty='None', solver='newton-cg'`) [56].

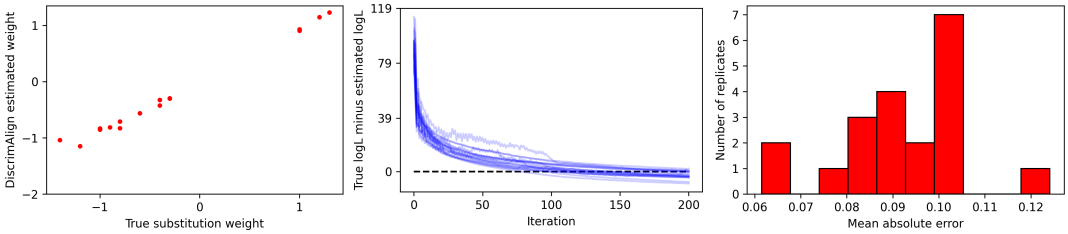



**Figure 2 Accuracy of estimation in replicated experiments on RNA sequences with simulated labels.** Left: Estimated vs ground-truth substitution weights in an example run. Middle: Trajectories of the difference between current log-likelihood and the ground-truth log-likelihood for 20 replicated experiments. Right: A histogram of mean absolute errors of estimation for 20 replicated experiments. All plots correspond to a model with an asymmetric substitution matrix and an affine gap penalty.

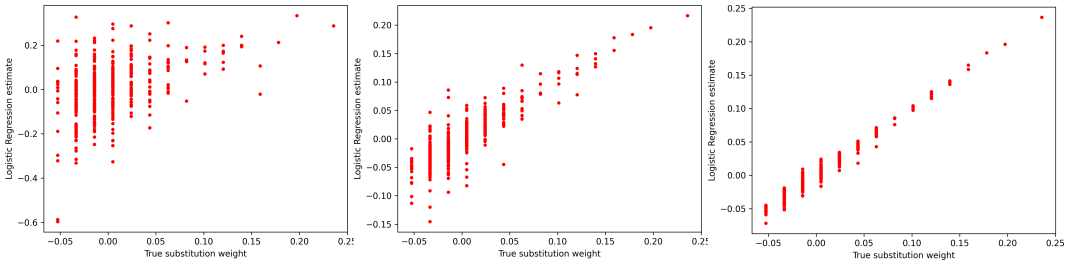



**Figure 3 A lower bound on the estimation error for asymmetric amino acid substitution matrices for different dataset sizes.** Logistic regression estimation of amino acid substitution matrices directly from optimal alignments obtained with the true parameters (without DiscrimAlign optimization). From left to right, $|\mathcal{D}| = 10^4, 10^5, 10^6$.

The large dataset $|\mathcal{D}| = 10^6$ resulted in a clear correlation between the estimated and ground-truth substitution weights (Fig 3), providing a likely indication of the minimal requirements for an accurate estimation of discriminative substitution matrices for detecting homology between *Homo sapiens* and *Gallus gallus* proteins. For $|\mathcal{D}| = 10^4$, the estimated parameters were only loosely correlated with the ground-truth, which is to be expected considering the large number of parameters. This correlation depended not just on $|\mathcal{D}|$, but also on the scaling of the substitution matrix; in particular, small parameter values were more difficult to estimate because of a lower impact on the probability (Fig. 3). A realistic range of substitution weights for amino acid discriminative substitution matrices requires further studies, and most likely depends on the similarity of compared proteins. The construction of proper datasets for estimation of discriminative matrices for homology search also requires further studies, as selecting positive and negative instances simply as related and unrelated proteins results in a complete separation. This can be overcome either by a specific selection of the dataset or developing regularized models.

## 4.3 Case study: prediction of miRNA-target interactions

The task of predicting miRNA target interactions is typically addressed with black-box methods such as convolutional neural networks [61], although approaches based on alignments with position-specific substitution matrices have been explored with promising results [67]. In this case study, we investigated whether a pairwise alignment score with a discriminative substitution matrix can be used to identify interacting pairs. We used the $\mathcal{D}_{\text{Hejret}}$ dataset

**Table 1** The area under the precision-recall curve (AUPRC) and final log-likelihood of miRNA interaction prediction for different alignment models trained with `DiscrimAlign`.

| Training dataset | Hejret Train | | | | Manakov Train | | | |
|---|---|---|---|---|---|---|---|---|
| Test dataset | Hejret Test | Manakov Test | Manakov Leave-Out | LogLik | Hejret Test | Manakov Test | Manakov Leave-Out | LogLik |
| Simple Linear | 0.852 | 0.746 | 0.741 | -3321.7 | 0.851 | 0.752 | 0.747 | -1219390.0 |
| Simple Affine | 0.848 | 0.761 | **0.757** | -3304.9 | 0.847 | 0.764 | 0.761 | -1202172.6 |
| Symmetric Linear | 0.852 | 0.754 | 0.751 | -3326.7 | **0.853** | 0.760 | 0.759 | -1208419.1 |
| Symmetric Affine | 0.849 | **0.761** | 0.756 | -3280.3 | 0.849 | 0.766 | 0.763 | -1199209.3 |
| Asymmetric Linear | **0.854** | 0.753 | 0.750 | -3302.6 | 0.852 | 0.761 | 0.759 | -1208102.0 |
| Asymmetric Affine | 0.852 | 0.760 | 0.756 | -3256.0 | 0.849 | **0.767** | **0.764** | -1197558.5 |

**Table 2** The Area under Precision-Recall curves for different miRNA-target prediction algorithms. Data for TargetScanCNN, miRBind and miRNA_CNN_Hejret2023 comes from ref. [61]. Full model refers to an asymmetric substitution matrix and affine gap penalty.

| Test dataset | Hejret Test | Manakov Test | Manakov Leave-Out |
|---|---|---|---|
| `TargetScanCNN` | 0.71 | 0.77 | 0.76 |
| `miRBind` | 0.80 | 0.71 | 0.71 |
| `miRNA_CNN_Hejret2023` | 0.77 | 0.71 | 0.71 |
| `miRNA_CNN_Hejret2023` retrained on Hejret Train | 0.86 | 0.79 | 0.78 |
| `miRNA_CNN_Hejret2023` retrained on Manakov Train | 0.84 | 0.84 | 0.81 |
| `DiscrimAlign` full model trained on Hejret Train | 0.85 | 0.76 | 0.76 |
| `DiscrimAlign` full model trained on Manakov Train | 0.85 | 0.77 | 0.76 |

and another, larger dataset from the miRBench package [61], referred to as Manakov, with 1 253 320 positive instances in the training split. This data set contains three splits: train, which we used to train our model; test, which we used to evaluate the model; and an additional left-out split which contains miRNAs not present in either the train or test split. We run DiscrimAlign on pairs of miRNA sequences and reverse-complemented target sequences with labels indicating experimentally confirmed interactions. We investigated all combinations of linear and affine gap penalties and simple, symmetric and asymmetric substitution matrices. We evaluated the performance using the area under the precision-recall curve (AUPRC) on the test split of each dataset in the same way as reported in the miRBench article [61].

For each training dataset, the training split was divided into fitting and validation subsets. The fitting subset was used to estimate alignment parameters for each candidate model, while the validation subset was used to select the best configuration. We considered local and global alignment, linear and affine gap penalties, and three substitution models: simple match/mismatch scores, symmetric substitution matrices, and general asymmetric substitution matrices. Performance was measured using the area under the precision-recall curve (AUPRC), while the final log-likelihood was used to assess model fit.

Asymmetric substitution matrices provided the best AUPRC when evaluated on the held-out test splits of their respective datasets, but not in cross-dataset evaluation, indicating some degree of overfitting to the experimental technique used in each dataset (Table 1). Models with a symmetric substitution matrix seemed to generalize better between datasets while having only slightly lower performance within datasets. Overall, models trained on the larger $\mathcal{D}_{\text{Manakov}}$ generalized well to the Hejret test set, achieving AUPRC values close to those of the Hejret-trained models, which was not the case for models trained on $\mathcal{D}_{\text{Hejret}}$ when evaluated on $\mathcal{D}_{\text{Manakov}}$. This suggests that the larger Manakov training set may capture broader miRNA-target pairing patterns.

Next, we refitted the full model (19 parameters) on full training splits and evaluated on the held-out test datasets (step size $\eta = 5 \times 10^{-4}$, 500 iterations). These models achieved AUPRC values comparable to neural-network baselines (Table 2). The substitution matrix obtained from the Hejret dataset was considerably asymmetric, likely reflecting the directional alignment of miRNA sequences to reverse-complemented target sequences.

$$M_{\text{Hejret}} = \left(\begin{array}{c|cccc} & A & C & T & G \\ \hline A & 0.66 & -0.60 & -0.45 & -0.60 \\ C & -0.72 & 0.65 & -0.77 & -0.86 \\ T & -0.57 & -0.36 & 0.62 & -0.58 \\ G & -0.36 & -0.78 & -0.58 & 0.63 \end{array}\right) \quad G_{\text{Hejret}} = \left(\begin{array}{c|c} \text{Gap open} & -1.12 \\ \text{Gap extend} & -0.11 \end{array}\right).$$

## 5 Discussion

Some of the first substitution matrices, both for amino acids and nucleotides, were derived from first principles, such as the physical properties of amino acids or the properties of the genetic code [55, 22, 36]. When databases of example alignments became available, researchers started to derive matrices that describe observed patterns of substitutions. In this work, we propose a new type of substitution matrices that we call *discriminative substitution matrices*, which describe the differences observed between two sets of pairs of sequences.

We described a general theoretical framework of learning discriminative matrices through a logistic link between the alignment score and the label associated with each pair. We analyzed the mathematical properties of alignment score and the logistic link as functions of substitution and gap weights. We derived subgradients of the alignment score with respect to the parameters and subgradients of the logistic link function and the log-likelihood. We used these results to implement a learning procedure and validated it on simulated data sets to show convergence and accuracy of estimation. Finally, we applied the method to learn a discriminative matrix for the task of predicting microRNA-target interactions. The discriminative alignment performed similarly to black-box classifiers based on neural networks while offering more biological insight into the interactions thanks to the interpretable nature of the model, which can be used in a downstream analysis to identify binding patterns by investigating the optimal alignments.

We remark that most amino acid substitution matrices are relatively similar, because certain pairs of amino acids align more commonly than others regardless of the data set (e.g. serine and threonine is more likely to align than serine and tryptophan, driving the differences in substitution weights). While discriminative matrices are thus not expected to be highly different than e.g. BLOSUM62 in terms of absolute differences of their values, even relatively minor changes can impact alignments and the preformance of database searches. A seemingly understudied problem is a quantitative analysis of sensitivity of alignments to parameter values in practical applications with commonly used substitution weights, which could answer how accurate the estimation of parameters needs to be.

The proposed model offers many topics for future investigations, some of which were mentioned throughout the paper. Other topics include improvements in computational efficiency and derivation of statistical tests for parameters, such as a likelihood ratio test to determine whether an affine gap penalty gives a statistically significant improvement in classification compared to a linear one in a particular application. Finally, while the proposed method is not explicitly designed to optimize the accuracy of alignments, it would be interesting to see how it performs compared to methods such as IPA or POP.

## References


1 Adelin Albert and John A Anderson. On the existence of maximum likelihood estimates in logistic regression models. *Biometrika*, 71(1):1–10, 1984.

2 Stephen F Altschul. Amino acid substitution matrices from an information theoretic perspective. *Journal of molecular biology*, 219(3):555–565, 1991.

3 Olivier Bastien, Sylvaine Roy, and Éric Maréchal. Construction of non-symmetric substitution matrices derived from proteomes with biased amino acid distributions. *Comptes rendus. Biologies*, 328(5):445–453, 2005.

4 Amir Beck. *First-order methods in optimization*. SIAM, 2017.

5 Kevin Brick and Elisabetta Pizzi. A novel series of compositionally biased substitution matrices for comparing plasmodium proteins. *BMC bioinformatics*, 9(1):236, 2008. doi:10.1186/1471-2105-9-236.

6 Celeste J Brown, Audra K Johnson, and Gary W Daughdrill. Comparing models of evolution for ordered and disordered proteins. *Molecular biology and evolution*, 27(3):609–621, 2010.

7 Christiam Camacho, George Coulouris, Vahram Avagyan, Ning Ma, Jason Papadopoulos, Kevin Bealer, and Thomas L Madden. Blast+: architecture and applications. *BMC bioinformatics*, 10(1):421, 2009. doi:10.1186/1471-2105-10-421.

8 Michał Aleksander Ciach, Elissavet Zacharopoulou, Michał Piotr Startek, Błażej Miasojedow, and Panagiotis Alexiou. DiscrimAlign. Software, version 1.0., swhId: swh:1:dir:e70ee75187e06eb93adc1da6022502a80230d4b0 (visited on 2026-08-12). URL: https://github.com/BioGeMT/DiscrimAlign, doi:10.4230/artifacts.27630.

9 Frank H Clarke. Generalized gradients and applications. *Transactions of the American Mathematical Society*, 205:247–262, 1975.

10 Frank H Clarke, Yu S Ledyaev, Ronald J Stern, and RR Wolenski. *Nonsmooth analysis and control theory*. Springer, 1998.

11 Peter JA Cock, Tiago Antao, Jeffrey T Chang, Brad A Chapman, Cymon J Cox, Andrew Dalke, Iddo Friedberg, Thomas Hamelryck, Frank Kauff, Bartek Wilczynski, et al. Biopython: freely available python tools for computational molecular biology and bioinformatics. *Bioinformatics*, 25(11):1422, 2009.

12 Cuong Cao Dang, Vincent Lefort, Vinh Sy Le, Quang Si Le, and Olivier Gascuel. Replacementmatrix: a web server for maximum-likelihood estimation of amino acid replacement rate matrices. *Bioinformatics*, 27(19):2758–2760, 2011. doi:10.1093/BIOINFORMATICS/BTR435.

13 Aris Daniilidis and Dmitriy Drusvyatskiy. Pathological subgradient dynamics. *SIAM Journal on Optimization*, 30(2):1327–1338, 2020. doi:10.1137/19M1298147.

14 Damek Davis, Dmitriy Drusvyatskiy, Sham Kakade, and Jason D Lee. Stochastic subgradient method converges on tame functions. *Foundations of computational mathematics*, 20(1):119–154, 2020. doi:10.1007/S10208-018-09409-5.

15 Margaret Dayhoff, R Schwartz, and B Orcutt. A model of evolutionary change in proteins. *Atlas of protein sequence and structure*, 5:345–352, 1978.

16 Richard Durbin, Sean R Eddy, Anders Krogh, and Graeme Mitchison. *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge university press, 1998. doi:10.1017/CBO9780511790492.

17 Robert C Edgar. Optimizing substitution matrix choice and gap parameters for sequence alignment. *Bmc Bioinformatics*, 10(1):396, 2009. doi:10.1186/1471-2105-10-396.

18 Robert C Edgar. Muscle5: High-accuracy alignment ensembles enable unbiased assessments of sequence homology and phylogeny. *Nature communications*, 13(1):6968, 2022.

19 Martin C Frith. How sequence alignment scores correspond to probability models. *Bioinformatics*, 36(2):408–415, 2020. doi:10.1093/BIOINFORMATICS/BTZ576.

20 Martin C Frith, Michiaki Hamada, and Paul Horton. Parameters for accurate genome alignment. *BMC bioinformatics*, 11(1):80, 2010. doi:10.1186/1471-2105-11-80.

21 Renganayaki Govindarajan, Biji Christopher Leela, and Achuthsankar S Nair. Rblosum performs better than corblosum with lesser error per query. *BMC Research Notes*, 11(1):328, 2018.

22 Richard Grantham. Amino acid difference formula to help explain protein evolution. *science*, 185(4154):862–864, 1974.

23 Jerrold R Griggs, Phil Hanlon, Andrew M Odlyzko, and Michael S Waterman. On the number of alignments of k sequences. *Graphs and Combinatorics*, 6(2):133–146, 1990. doi:10.1007/BF01787724.

24 Dan Gusfield and Paul Stelling. Parametric and inverse-parametric sequence alignment with xparal. In *Methods in enzymology*, volume 266, pages 481–494. Elsevier, 1996.

25 Michiaki Hamada, Yukiteru Ono, Kiyoshi Asai, and Martin C Frith. Training alignment parameters for arbitrary sequencers with last-train. *Bioinformatics*, 33(6):926–928, 2017. doi:10.1093/BIOINFORMATICS/BTW742.

26 Vaclav Hejret, Nandan Mysore Varadarajan, Eva Klimentova, Katarina Gresova, Ilektra-Chara Giassa, Stepanka Vanacova, and Panagiotis Alexiou. Analysis of chimeric reads characterises the diverse targetome of ago2-mediated regulation. *Scientific Reports*, 13(1):22895, 2023.

27 Steven Henikoff and Jorja G Henikoff. Amino acid substitution matrices from protein blocks. *Proceedings of the national academy of sciences*, 89(22):10915–10919, 1992.

28 Martin Hess, Frank Keul, Michael Goesele, and Kay Hamacher. Addressing inaccuracies in blosum computation improves homology search performance. *Bmc Bioinformatics*, 17(1):189, 2016. doi:10.1186/S12859-016-1060-3.

29 Reiner Horst and Nguyen V Thoai. Dc programming: overview. *Journal of Optimization Theory and Applications*, 103(1):1–43, 1999.

30 Hyeonseo Hwang, Hee Ryung Chang, and Daehyun Baek. Determinants of functional microrna targeting. *Molecules and cells*, 46(1):21–32, 2023.

31 Sumaiya Islam and Robert J Pantazes. Developing similarity matrices for antibody-protein binding interactions. *Plos one*, 18(10):e0293606, 2023.

32 Kejue Jia and Robert L Jernigan. New amino acid substitution matrix brings sequence alignments into agreement with structure matches. *Proteins: Structure, Function, and Bioinformatics*, 89(6):671–682, 2021.

33 Thorsten Joachims, Tamara Galor, and Ron Elber. Learning to align sequences: A maximum-margin approach. In *New algorithms for macromolecular simulation*, pages 57–69. Springer, 2005.

34 Kaisa Joki, Adil M Bagirov, Napsu Karmitsa, Marko M Makela, and Sona Taheri. Double bundle method for finding clarke stationary points in nonsmooth dc programming. *SIAM Journal on Optimization*, 28(2):1892–1919, 2018. doi:10.1137/16M1115733.

35 David T Jones, William R Taylor, and Janet M Thornton. The rapid generation of mutation data matrices from protein sequences. *Bioinformatics*, 8(3):275–282, 1992. doi:10.1093/BIOINFORMATICS/8.3.275.

36 Samuel Karlin and Stephen F Altschul. Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. *Proceedings of the National Academy of Sciences*, 87(6):2264–2268, 1990.

37 Kazutaka Katoh, John Rozewicki, and Kazunori D Yamada. Mafft online service: multiple sequence alignment, interactive sequence choice and visualization. *Briefings in bioinformatics*, 20(4):1160–1166, 2019. doi:10.1093/BIB/BBX108.

38 John Kececioglu and Eagu Kim. Simple and fast inverse alignment. In *Annual International Conference on Research in Computational Molecular Biology*, pages 441–455. Springer, 2006. doi:10.1007/11732990_37.

39 Frank Keul, Martin Hess, Michael Goesele, and Kay Hamacher. Pfasum: a substitution matrix from pfam structural alignments. *BMC bioinformatics*, 18(1):293, 2017. doi:10.1186/S12859-017-1703-Z.

40 Eagu Kim and John Kececioglu. Learning scoring schemes for sequence alignment from partial examples. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 5(4):546–556, 2008. doi:10.1109/TCBB.2008.57.

41 Satoshi Koide, Keisuke Kawano, and Takuro Kutsuna. Neural edit operations for biological sequences. *Advances in Neural Information Processing Systems*, 31, 2018.

42 Carolin Kosiol and Nick Goldman. Different versions of the dayhoff rate matrix. *Molecular biology and evolution*, 22(2):193–199, 2005.

43 Jacek Krol, Inga Loedige, and Witold Filipowicz. The widespread regulation of microrna biogenesis, function and decay. *Nature reviews genetics*, 11(9):597–610, 2010.

44 Igor B Kuznetsov. Protein sequence alignment with family-specific amino acid similarity matrices. *BMC Research Notes*, 4(1):296, 2011.

45 Si Quang Le and Olivier Gascuel. An improved general amino acid replacement matrix. *Molecular biology and evolution*, 25(7):1307–1320, 2008.

46 Claire Lemaitre, Aurélien Barré, Christine Citti, Florence Tardy, François Thiaucourt, Pascal Sirand-Pugnet, and Patricia Thébault. A novel substitution matrix fitted to the compositional bias in mollicutes improves the prediction of homologous relationships. *BMC bioinformatics*, 12(1):457, 2011. doi:10.1186/1471-2105-12-457.

47 Sean E McGeary, Kathy S Lin, Charlie Y Shi, Thy M Pham, Namita Bisaria, Gina M Kelley, and David P Bartel. The biochemical basis of microrna targeting efficacy. *Science*, 366(6472):eaav1741, 2019.

48 Uros Midic, A Keith Dunker, and Zoran Obradovic. Protein sequence alignment and structural disorder: a substitution matrix for an extended alphabet. In *Proceedings of the KDD-09 Workshop on Statistical and Relational Learning in Bioinformatics*, pages 27–31, 2009. doi:10.1145/1562090.1562096.

49 James T Morton, Charlie EM Strauss, Robert Blackwell, Daniel Berenberg, Vladimir Gligorijevic, and Richard Bonneau. Protein structural alignments from sequence. *BioRxiv*, pages 2020–11, 2020.

50 Tobias Müller, Rainer Spang, and Martin Vingron. Estimating amino acid substitution models: a comparison of dayhoff's estimator, the resolvent approach and a maximum likelihood method. *Molecular biology and evolution*, 19(1):8–13, 2002.

51 Tobias Müller and Martin Vingron. Modeling amino acid replacement. *Journal of Computational Biology*, 7(6):761–776, 2000. doi:10.1089/10665270050514918.

52 VN Novoseletsky, GA Armeev, and KV Shaitan. Construction and analysis of amino acid substitution matrices for optimal alignment of microbial rhodopsin sequences. *Moscow University Biological Sciences Bulletin*, 74(1):21–25, 2019.

53 Evgenii Ofitserov, Vasily Tsvetkov, and Vadim Nazarov. Soft edit distance for differentiable comparison of symbolic sequences. *arXiv preprint arXiv:1904.12562*, 2019. arXiv:1904.12562.

54 Lior Pachter and Bernd Sturmfels. *Algebraic statistics for computational biology*, volume 13. Cambridge university press, 2005.

55 William R Pearson. Selecting the right similarity-scoring matrix. *Current protocols in bioinformatics*, 43(1):3–5, 2013.

56 F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. doi:10.5555/1953048.2078195.

57 Samantha Petti, Nicholas Bhattacharya, Roshan Rao, Justas Dauparas, Neil Thomas, Juannan Zhou, Alexander M Rush, Peter Koo, and Sergey Ovchinnikov. End-to-end learning of multiple sequence alignments with differentiable smith–waterman. *Bioinformatics*, 39(1):btac724, 2023. doi:10.1093/BIOINFORMATICS/BTAC724.

58 Valery Polyanovsky, Alexander Lifanov, Natalia Esipova, and Vladimir Tumanyan. The ranging of amino acids substitution matrices of various types in accordance with the alignment accuracy criterion. *BMC bioinformatics*, 21(Suppl 11):294, 2020. doi:10.1186/S12859-020-03616-0.

59 Predrag Radivojac, Zoran Obradovic, Celeste J Brown, and A Keith Dunker. Improving sequence alignments for intrinsically disordered proteins. In *Biocomputing 2002*, pages 589–600. World Scientific, 2001.

60 Rodolfo Rios-Zertuche. Examples of pathological dynamics of the subgradient method for lipschitz path-differentiable functions. *Mathematics of Operations Research*, 47(4):3184–3206, 2022. doi:10.1287/MOOR.2021.1241.

61 Stephanie Sammut, Katarina Gresova, Dimosthenis Tzimotoudis, Eva Marsalkova, David Cechak, and Panagiotis Alexiou. mirbench: novel benchmark datasets for microrna binding site prediction that mitigate against prevalent microrna frequency class bias. *Bioinformatics*, 41(Supplement_1):i542–i551, 2025. doi:10.1093/BIOINFORMATICS/BTAF233.

62 Eric W Sayers, Jeffrey Beck, Evan E Bolton, J Rodney Brister, Jessica Chan, Ryan Connor, Michael Feldgarden, Anna M Fine, Kathryn Funk, Jinna Hoffman, et al. Database resources of the national center for biotechnology information in 2025. *Nucleic acids research*, 53(D1):D20–D29, 2025.

63 David J States, Warren Gish, and Stephen F Altschul. Improved sensitivity of nucleic acid database searches using application-specific scoring matrices. *Methods*, 3(1):66–70, 1991.

64 Martin Steinegger and Johannes Söding. Mmseqs2 enables sensitive protein sequence searching for the analysis of massive data sets. *Nature biotechnology*, 35(11):1026–1028, 2017.

65 Mark P Styczynski, Kyle L Jensen, Isidore Rigoutsos, and Gregory Stephanopoulos. Blosum62 miscalculations improve search performance. *Nature biotechnology*, 26(3):274–275, 2008.

66 Fangting Sun, David Fernández-Baca, and Wei Yu. Inverse parametric sequence alignment. *Journal of Algorithms*, 53(1):36–54, 2004. doi:10.1016/J.JALGOR.2004.04.008.

67 Amlan Talukder, Xiaoman Li, and Haiyan Hu. Position-wise binding preference is important for mirna target site prediction. *Bioinformatics*, 36(12):3680–3686, 2020. doi:10.1093/BIOINFORMATICS/BTAA195.

68 Octavian Teodorescu, Tamara Galor, Jaroslaw Pillardy, and Ron Elber. Enriching the sequence substitution matrix by structural information. *Proteins: Structure, Function, and Bioinformatics*, 54(1):41–48, 2004.

69 Rakesh Trivedi and Hampapathalu Adimurthy Nagarajaram. Substitution scoring matrices for proteins-an overview. *Protein Science*, 29(11):2150–2163, 2020.

70 Simon Whelan and Nick Goldman. A general empirical model of protein evolution derived from multiple protein families using a maximum-likelihood approach. *Molecular biology and evolution*, 18(5):691–699, 2001.

71 Alex J Wilkie. Model completeness results for expansions of the ordered field of real numbers by restricted pfaffian functions and the exponential function. *Journal of the American Mathematical Society*, 9(4):1051–1094, 1996.

72 Kazunori Yamada and Kentaro Tomii. Revisiting amino acid substitution matrices for identifying distantly related proteins. *Bioinformatics*, 30(3):317–325, 2014. doi:10.1093/BIOINFORMATICS/BTT694.

73 Yin Yao and Martin C Frith. Improved dna-versus-protein homology search for protein fossils. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 20(3):1691–1699, 2022. doi:10.1109/TCBB.2022.3177855.


# A Proofs

**Proof of Proposition 1.** Using Equation (2), we can express $U_{A,B}(x)$ as

$$U_{A,B}(x^*) = \{\theta \in \Theta \colon \forall_{x \in \mathcal{X}_{A,B}} \langle \theta, x^* \rangle \geq \langle \theta, x \rangle\}.$$

From the linearity of the inner product, $\langle \theta, x^* \rangle \geq \langle \theta, x \rangle$ is equivalent to $\langle \theta, x^* - x \rangle \geq 0$. This allows us to write

$$U_{A,B}(x^*) = \{\theta \in \Theta \colon \forall_{x \in \mathcal{X}_{A,B}} \langle \theta, x^* - x \rangle \geq 0\} = \bigcap_{x \in \mathcal{X}_{A,B}} \{\theta \in \Theta \colon \langle \theta, x^* - x \rangle \geq 0\}.$$

For any vector $y$, the set defined by the inequality $\langle \theta, y \rangle \geq 0$ is a closed half-space, which is a convex set. Therefore, $U_{A,B}(x)$ is a closed convex set as an intersection of a finite number of such sets. Furthermore, since for any scalar $s$ we have $\langle s\theta, y \rangle = s\langle \theta, y \rangle$, it follows that for $s > 0$ we have $\langle s\theta, y \rangle \geq 0$ if and only if $\langle \theta, y \rangle \geq 0$, which shows that $U_{A,B}(x)$ is a cone. The zero vector of parameters always belongs to $U_{A,B}(x)$, because $\langle 0, y \rangle = 0$ for any vector $y$. The proof for $U^*_{A,B}(x)$ is similar; however, this set is open as an intersection of a finite number of open half-spaces. Furthermore, the zero vector never belongs to $U^*_{A,B}(x)$, and the set may be empty. ◀

**Proof of Lemma 2.** For the proof of Lipschitz continuity, consider two vectors of parameters $\theta$ and $\theta'$. First, we note that

$$\max_{x \in \mathcal{X}_{A,B}} \langle \theta, x \rangle - \max_{x \in \mathcal{X}_{A,B}} \langle \theta', x \rangle \leq \max_{x \in \mathcal{X}_{A,B}} (\langle \theta, x \rangle - \langle \theta', x \rangle) = \max_{x \in \mathcal{X}_{A,B}} \langle \theta - \theta', x \rangle.$$

The inequality follows from the fact that

$$\begin{aligned} \max_{x \in \mathcal{X}_{A,B}} \langle \theta, x \rangle &= \max_{x \in \mathcal{X}_{A,B}} \langle (\theta - \theta') + \theta', x \rangle = \\ &= \max_{x \in \mathcal{X}_{A,B}} (\langle \theta - \theta', x \rangle + \langle \theta', x \rangle) \leq \max_{x \in \mathcal{X}_{A,B}} \langle \theta - \theta', x \rangle + \max_{x \in \mathcal{X}_{A,B}} \langle \theta', x \rangle. \end{aligned}$$

It follows that $S_\theta(A,B) - S_{\theta'}(A,B) \leq S_{\theta-\theta'}(A,B)$. Next, using the Cauchy-Schwarz inequality $\langle a, b \rangle \leq ||a|| \cdot ||b||$, we have

$$S_{\theta-\theta'}(A,B) = \max_{x \in \mathcal{X}_{A,B}} \langle \theta - \theta', x \rangle \leq \max_{x \in \mathcal{X}_{A,B}} (||\theta - \theta'|| \cdot ||x||) = ||\theta - \theta'|| \max_{x \in \mathcal{X}_{A,B}} ||x||.$$

We therefore have $S_\theta(A,B) - S_{\theta'}(A,B) \leq ||\theta - \theta'|| \max_{x \in \mathcal{X}_{A,B}} ||x||$. Swapping $\theta$ and $\theta'$ (and noting that $||\theta - \theta'|| = ||\theta' = \theta||$), we obtain $S_{\theta'}(A,B) - S_\theta(A,B) \leq ||\theta - \theta'|| \max_{x \in \mathcal{X}_{A,B}} ||x||$, and thus we conclude that

$$|S_\theta(A,B) - S_{\theta'}(A,B)| \leq ||\theta - \theta'|| \max_{x \in \mathcal{X}_{A,B}} ||x||.$$

For the proof of convexity, consider $p \in [0,1]$ and two vectors of parameters $\theta$ and $\theta'$. Then,

$$\begin{aligned} S_{p\theta+(1-p)\theta'}(A,B) &= \max_{x \in X_{A,B}} \langle p\theta + (1-p)\theta', x \rangle = \\ &= \max_{x \in X_{A,B}} (p\langle \theta, x \rangle + (1-p)\langle \theta', x \rangle) \leq \\ &\leq p \max_{x \in X_{A,B}} \langle \theta, x \rangle + (1-p) \max_{x \in X_{A,B}} \langle \theta', x \rangle = \\ &= pS_\theta(A,B) + (1-p)S_{\theta'}(A,B) \end{aligned}$$ ◀

**Proof of Proposition 5.** Since the functions $\theta \to S_\theta(A_i, B_i)$ is differentiable almost everywhere, so is $l(\alpha, \theta; \mathcal{D})$. Accordingly, let $\theta$ be such that every pair of sequences $A_i, B_i$ has a unique optimal configuration $x_i$, and let $U = \bigcap_i U^*(x_i)$ be the set of $\theta$ where all of these configurations are uniquely optimal. Then, the log-likelihood restricted to $U$ can be expressed as

$$\begin{aligned} l(\alpha, \theta; \mathcal{D})_{|\theta \in U} = &\sum_{(A_i,B_i,y_i) \in \mathcal{D}} y_i(\alpha + \langle \theta, x_i \rangle) \\ &- \sum_{(A_i,B_i,y_i) \in \mathcal{D}} \ln\left(1 + \exp(\alpha + \langle \theta, x_i \rangle)\right). \end{aligned}$$

Now, $\sum_{(A_i,B_i,y_i) \in \mathcal{D}} y_i(\alpha + \langle \theta, x_i \rangle)$ is linear with respect to $\theta$, making it both convex and concave. Therefore, $l(\alpha, \theta; \mathcal{D})_{|\theta \in U}$ is concave with respect to $\theta$ as a sum of concave functions. ◀

**Proof of Propostition 10.** First, we observe that the functions $\ln p(\alpha, \theta; A_i, B_i)$, $\ln(1 - p(\alpha, \theta; A_i, B_i))$ are locally Lipschitz continuous with respect to $\theta$, so that their Clarke subdifferentials are well-defined. For a fixed intercept $\alpha$, let $\varsigma(x; \alpha) \colon \mathbb{R} \to \mathbb{R}$ be the logistic function of $x$ defined as $\varsigma(x; \alpha) = 1/(1 + \exp(-\alpha - x))$. This function is globally Lipschitz continuous with respect to $x$, because for any $x \in \mathbb{R}$ we have $\varsigma'(x) = \varsigma(x)(1 - \varsigma(x)) < 1$. Next, since $p(\alpha, \theta; A_i, B_i) = \varsigma(S_\theta(A_i, B_i); \alpha)$, and a composition of Lipschitz functions is Lipschitz, it follows that $p(\alpha, \theta; A_i, B_i)$ is also globally Lipschitz continuous with respect to $\theta$. Finally, $\ln(x)$ is locally (but not globally) Lipschitz continuous. The proof for $\ln(1 - p(\theta))$ follows the same lines. We remark that the function $l(\alpha, \theta)$ is also locally Lipschitz with respect to $\theta$ as a sum of locally Lipschitz functions.

Now, the first equality in the Proposition follows directly from the chain rule in Lemma 9 applied to $f(\theta) = S_\theta(A, B)$ and $g_\alpha(x) = -\ln(1 + \exp(-\alpha - x))$ for a fixed $\alpha$. The second equality follows from the chain rule applied to $g_\alpha(x) = -\ln(1 + \exp(\alpha + x))$. ◀

**Proof of Lemma 11.** First, we note that from the definition of the Clarke subdifferential, we have $\partial^C_\theta l(\alpha, \theta; \mathcal{D}) = \mathrm{Conv}(\{v_i\})$ for a set of vectors $v_i$, such that for each $v_i$ there exists a sequence $\theta^i_n \to \theta$ such that $l(\alpha, \theta; \mathcal{D})$ is differentiable w.r.t. $\theta$ at each $\theta_n$ and $\nabla_\theta l(\alpha, \theta^i_n; \mathcal{D}) \to v_i$.

Assume $\theta_n \to \theta$ is such that $\mathcal{B}(\theta_n; \mathcal{D}) = \{(x_1, \ldots, x_{|\mathcal{D}|})\}$ for all $n$. Then, for every pair of sequences $A_i, B_i$, the score $S_{\theta_n}(A_i, B_i)$ is differentiable w.r.t. $\theta$ at each $\theta_n$ (as the alignment configurations are unique). It follows that $\nabla_\theta l(\alpha, \theta_n; \mathcal{D})$ exists at all $\theta_n$. From Proposition 10, we get

$$\begin{aligned}\nabla_\theta l(\alpha, \theta_n; \mathcal{D}) &= \sum_{(A_i, B_i, y_i) \in \mathcal{D}} y_i(1 - p(\theta_n))x_i + (1 - y_i)(-p(\theta_n))x_i \\ &= \sum_{(A_i, B_i, y_i) \in \mathcal{D}} (y_i - p(\theta_n))x_i.\end{aligned}$$

From the continuity of $p$ it follows that $\nabla_\theta l(\alpha, \theta_n; \mathcal{D}) \to \sum_{(A_i, B_i, y_i) \in \mathcal{D}} (y_i - p(\theta))x_i$, and thus $\sum_{(A_i, B_i, y_i) \in \mathcal{D}} (y_i - p(\theta))x_i \in \partial^C_\theta l(\alpha, \theta_n; \mathcal{D})$.

We remark that the reason why the Lemma states an inclusion, rather than equality, is that a sum of non-differentiable functions can be differentiable (e.g. $f(x) = |x|$ and $g(x) = -|x|$). Therefore, even if the alignment scores are not differentiable at points of some sequence $\theta_n \to \theta$, the log-likelihood potentially may be differentiable at these points and result in additional limits of gradients belonging to the Clarke subdifferential. Whether cases like that can happen in alignments remains an open question. ◀

**Proof of Theorem 14.** First, we note that the log-likelihood function $l(\alpha, \theta; \mathcal{D})$ is a "tame" function in the sense of Davis, Drusvyatsky, Kakade and Lee and satisfies their Assumption D [14]. This follows from the fact that the function is composed of a finite number of algebraic operations, maxima over finite numbers of functions, and logarithmic and exponential functions, which makes it definable in the o-minimal structure $\mathbb{R}_{\exp} = (\mathbb{R}, +, ., \exp, 0, 1)$ [71].

We will now show that the procedure satisfies Assumption C form [14]. The assumptions $\eta_k > 0, \sum \eta_k = \infty, \sum \eta_k^2 < \infty$ correspond to Assumption C.1, and the assumption that $\theta_k$ is bounded corresponds to Assumption C.2. Now, let $\alpha_k = \xi_k/\eta_k$, so that the update step can be written as $\theta_{k+1} = \theta_k + \eta_k(g_k + \alpha_k)$. Now, since $\mathbb{E}\xi_k = 0$, we also have $\mathbb{E}\alpha_k = 0$. Furthermore, $\mathbb{E}||\alpha_k||^2 = \frac{1}{\eta_k^2}\mathbb{E}||\xi_k||^2 = n\sigma_k^2/\eta_k^2$. Therefore, $\alpha_k$ is a sequence of independent random variables with zero expected value and a bounded variance, which satisfies Assumption C.3. ◀

**Proof of Theorem 15.** Let $g_k \in \partial_\theta^C l(\alpha, \theta; \mathcal{D}_k)$. Let $I_1^k, I_2^k, \ldots, I_{|\mathcal{D}|}^k$ be a sequence of independent binary random variables such that $I_i^k = 1$ if the $i$-th pair was sampled in iteration $k$. By construction of $\mathcal{D}_k$, we have $\mathbb{P}(I_i^k = 1) = N/|\mathcal{D}|$.

Since the Theorem assumes that $g_k$ is constructed using Proposition 12, conditioned on the sequence $I^k = (I_1^k, \ldots, I_{|\mathcal{D}|}^k)$ it can be expressed as

$$g_k | I^k = \sum_{i=1}^{|\mathcal{D}|} I_i^k (y_i - p(\alpha, \theta; A_i, B_i)) x_i.$$

Let $\tilde{g}_k$ be a hypothetical subgradient obtained using Proposition 12 on the full dataset,

$$\tilde{g}_k = \sum_{i=1}^{|\mathcal{D}|} (y_i - p(\alpha, \theta; A_i, B_i)) x_i.$$

We now have

$$\mathbb{E} g_k = \frac{N}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} (y_i - p(\alpha, \theta; A_i, B_i)) x_i = \frac{N}{|\mathcal{D}|} \tilde{g}_k.$$

Let $\alpha_k = \xi_k / \eta_k$ and let $\beta_k = \tilde{g}_k - |\mathcal{D}|(g_k + \alpha_k)/N$. We can now write the $\theta$ update step as

$$\theta_{k+1} = \theta_k + \frac{N}{|\mathcal{D}|} \eta_k (\tilde{g}_k - \beta_k).$$

The Theorem now follows from Theorem 14 applied to the above sequence, treating $N\eta_k/|\mathcal{D}|$ as the sequence of step lengths and $\beta_k$ as the sequence of random errors. ◀

# Fast Set Operations for Compact $k$-mer Sets

**Jarno N. Alanko**
Department of Computer Science, University of Helsinki, Finland

**Lore Depuydt**
Univ. Lille, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France

**Camille Marchet**
Univ. Lille, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France

**Simon J. Puglisi**
Department of Computer Science, University of Helsinki, Finland

## Abstract

The $k$-mer spectrum of a set of sequences is the set of $k$-length substrings the sequences contain. This lossy representation of sequence content pervades modern genomics. Recently, the spectral Burrows-Wheeler transform (SBWT) has emerged as a space-efficient representation of $k$-spectra that also supports efficient $k$-mer lookup queries and, more generally, easy navigation of the de Bruijn graph of the $k$-spectrum. In this paper, we examine primitive set operations, such as intersection, union, and set difference, on SBWT-encoded $k$-spectra and show that these operations can be supported efficiently. Moreover, efficient merging leads directly to a new memory-efficient algorithm for SBWT construction, which was able to build the SBWT for the 661K bacterial dataset containing 88 billion distinct $k$-mers in 50 hours using 186 GiB of RAM and 112 GiB of disk space. Given the pervasiveness of $k$-mer sets in genomics and the continued rapid growth of genomic databases, our work opens the door to a wide array of future applications that manipulate and reason about genomic data by dealing directly with simultaneously compact and searchable $k$-mer set representations offered by the SBWT.

**2012 ACM Subject Classification** Theory of computation → Design and analysis of algorithms

**Keywords and phrases** Data Structures, efficient Algorithms

**Digital Object Identifier** 10.4230/LIPIcs.WABI.2026.18

**Related Version** *Preprint*: `https://doi.org/10.64898/2026.05.24.727514`

**Supplementary Material** *Software*: `https://github.com/LoreDepuydt/sbwt-set-operations` [4] archived at `swh:1:dir:de3c949dbd910f97f6af25f8be4838be2fbe4dfb`

**Funding** This work has benefited from funding from the French State under the France 2030 program, reference ANR-21-IDES-0006. The European Metropolis of Lille and the University of Lille are also acknowledged for their funding and support of the project WILL-CHAIRES-25-001-BOSSA.

## 1 Introduction

The $k$-mer spectrum of a set of sequences is the set of $k$-length substrings that the sequences contain and is a lossy representation of sequence content that pervades modern genomics. $k$-mers are amenable to hashing, counting, and indexing at speeds that alignment-based approaches cannot match, and they are present in a wide range of applications from genome assembly to taxonomic classification. In rough terms, $k$-mers let pipelines scale by sacrificing an acceptable loss of fidelity (long-range relationships) for tractability.

This scalability is especially important given the recent emergence of peta-scale $k$-mer indexes over public sequence collections [8, 13], which allow researchers to query the entirety of deposited sequencing data in minutes. However, an open problem is that these systems


© Jarno N. Alanko, Lore Depuydt, Camille Marchet, and Simon J. Puglisi;
licensed under Creative Commons License CC-BY 4.0



are largely static, updates require costly rebuilds, and supported queries are narrow. As repositories such as AllTheBacteria [12] and the Human Pangenome Reference Consortium [20] regularly release new assemblies, freeze-and-rebuild will become unsustainable.

Beyond the index update problem, biologically meaningful questions – essentially questions of data mining – such as determining the set of $k$-mers shared across a panel of conditions [9], require boolean set operations over indexed $k$-mer sets. This type of query is not yet supported in large scale systems, as in-place set operations on compact $k$-mer representations remain both time- and space-intensive [15].

**Contribution.** An ideal $k$-mer representation is simultaneously (i) compact, fitting large collections in modest hardware; (ii) searchable, supporting fast membership and navigation queries; and (iii) composable, allowing indexes to be combined via union, intersection, and difference without rebuilding from raw sequence. The first two are met by the spectral Burrows-Wheeler transform (SBWT) [6], which represents $k$-spectra space-efficiently while supporting $k$-mer lookup and de Bruijn graph navigation in time independent of set size.

In this paper, we address the third problem above by giving union, intersection, and set difference on SBWT-encoded $k$-spectra, built around a compact $\mathcal{O}(n)$-bit merge plan constructible in $\mathcal{O}(nk)$ time. As a direct application, we build an SBWT index over the full 661K bacterial dataset [7], consisting of 2.57 Tbp and 88.70 billion distinct 31-mers, in 50 hours using 186 GiB peak RAM and 112 GiB peak disk space, where direct construction would need an estimated 1.2 TiB of RAM or 5.6 TiB of disk space.

**Related work.** Set operations on $k$-mer collections have been approached from several angles. KMC [14] stores $k$-mers in sorted, partitioned databases and supports union, intersection, and subtraction via streaming merges; CBL [15] provides the same three operations on compact in-memory $k$-mer sets, and de Bruijn graph indexes such as Bifrost [11] and MetaGraph [13] expose merge operations on their respective representations. Minimal perfect hashing approaches (e.g., SSHash [18]) offer fast queries, but combining two indexes requires rebuilding the $k$-mer representation and hash function from scratch. On the indexed-Burrows-Wheeler side, Muggli et al. [17] give an $\mathcal{O}(nk)$-time algorithm for merging BOSS representations of de Bruijn graphs, and Egidi et al. [10] describe an $\mathcal{O}(n^2)$-time algorithm for general Wheeler graphs together with a space-improved $\mathcal{O}(nk)$ variant for BOSS. Alanko et al. [1] support dynamic $k$-mer sets via an $\mathcal{O}(nk)$ merge on a Wheeler index of a de Bruijn graph, simulating deletions with a bit vector.

**Paper Outline.** Section 2 fixes notation for $k$-spectra and the SBWT. Section 3.1 introduces the *merge plan*, a compact $\mathcal{O}(n)$-bit encoding of how two padded $k$-spectra interleave in colexicographic order, and gives an $\mathcal{O}(nk)$-time construction algorithm together with a sketch of an $\mathcal{O}(n \log k)$ variant. In Section 3.2, we use the merge plan to construct functional SBWT indexes of the union, intersection, and set difference. Section 4 evaluates the resulting algorithms against KMC and a sorted-list baseline, and applies the merge operation to build an SBWT over the full 661K bacterial dataset.

## 2 Preliminaries

Throughout this paper, a *string* $X[1..n]$ is a sequence of $|X|$ symbols over the DNA alphabet $\Sigma = \{\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{T}\}$. The empty string is denoted $\epsilon$ and $|\epsilon| = 0$. The *substring* of $X$ starting at symbol $i$ and ending at symbol $j$ is denoted $X[i..j]$, which equals $\epsilon$ if $i > j$. We also use the half-open interval notations $X(i..j] = X[i+1..j]$ and $X[i..j) = X[i..j-1]$. A *prefix* is a substring with starting position 1 and a *suffix* is a substring with ending position $n$.

A *k-mer* refers to a (sub)string of length $k$. The set of distinct $k$-mers of a string $X$ is called the *k-spectrum* of $X$.

▶ **Definition 1** ($k$-Spectrum). *The k-spectrum $S_k(X)$ of a string $X$ is the set of distinct k-mers $\{X[i..i+k-1] \mid i = 1, \ldots, |X| - k + 1\}$.*

▶ **Definition 2** ($k$-Spectrum of a set of strings). *The k-spectrum $S_k(X_1, \ldots, X_m)$ of a set of strings $X_1, \ldots, X_m$ is the union $\bigcup_{i=1}^{m} S_k(X_i)$.*

A *padded k-spectrum* adds a minimal set of \$-padded *dummy* $k$-mers to ensure that in the de Bruijn graph of the $k$-mers, every non-dummy $k$-mer has an incoming path of length at least $k$:

▶ **Definition 3** (Padded $k$-Spectrum). *Let $R = S_k(X_1, \ldots, X_m)$ be a k-spectrum with alphabet $\Sigma$, and let $R' \subseteq R$ be the set of k-mers $Y$ such that $Y[1..k-1]$ is not a suffix of any k-mer in $R$. The padded k-spectrum is the set $S_k^+(X_1, \ldots, X_m) = R \cup \{\$^k\} \cup \bigcup_{Y \in R'} \{\$^{k-i} Y[1..i] \mid i = 1, \ldots, k-1\}$, with special character $\$ \notin \Sigma$ and $\$ < c$ for all $c \in \Sigma$.*

For example, if $X_1 = \mathsf{ACGT}$, $X_2 = \mathsf{GACG}$ and $k = 3$, then $S_3(X_1, X_2) = \{\mathsf{ACG}, \mathsf{CGT}, \mathsf{GAC}\}$ and $S_3^+(X_1, X_2) = \{\mathsf{ACG}, \mathsf{CGT}, \mathsf{GAC}, \$\$\$, \$\$\mathsf{G}, \$\mathsf{GA}\}$. The $k$-mers that contain the dollar-symbols are called *dummy k-mers*.

The spectral Burrows-Wheeler transform maps a padded $k$-spectrum into a sequence of subsets of the alphabet:

▶ **Definition 4** (Spectral Burrows-Wheeler transform (SBWT)). *Let $R^+$ be a padded k-spectrum and let $X_1, \ldots, X_{|R^+|}$ be the elements of $R^+$ in colexicographic order (i.e., comparing the k-mers from right to left). The SBWT is the sequence of sets $A_1, \ldots, A_{|R^+|}$ with $A_i \subseteq \Sigma$ such that $A_i = \emptyset$ if $i > 1$ and $X_i[2..k] = X_{i-1}[2..k]$, otherwise $A_i = \{c \in \Sigma \mid X_i[2..k]c \in R^+\}$.*

It is helpful to think of the SBWT as encoding the set $R^+$ of padded $k$-mers as a list arranged in colexicographic order. The *SBWT graph* is a subgraph of the node-centric de Bruijn graph of the $k$-mers, which contains only the smallest incoming de Bruijn graph edge into each node, where edges are compared by the colexicographic order of their origin nodes. This graph gives an alternative and equivalent way of defining the SBWT subset sequence: the $i$-th set in the SBWT sequence is the set of outgoing edge labels in the SBWT graph from the $i$-th $k$-mer in colexicographic order. We refer the reader to prior work [6, 3, 2] for examples and a more in-depth discussion of the SBWT.

In practice, the SBWT is represented as a bit matrix with one column $\mathrm{col}_c$ per character $c \in \Sigma$, where row $p$ has a set bit in column $\mathrm{col}_c$ if $c \in A_p$. From now on, we write $\mathrm{SBWT}[p]$ for the set $A_p$. Let $C[c]$ be the number of edge labels smaller than $c$ in the SBWT graph. For a row $p$ and character $c \in \mathrm{SBWT}[p]$, let $\mathrm{succ}(p, c)$ denote the destination node reached by following the outgoing edge labelled $c$ from row $p$. This operation is supported by the default SBWT representation [6]: $\mathrm{succ}(p, c) = 1 + C[c] + \mathrm{subsetrank}_c(p)$, where $\mathrm{subsetrank}_c(p)$ is the number of rows $j \leq p$ such that $c \in \mathrm{SBWT}[j]$. The $+1$ skips the source node corresponding to $k$-mer $\$^k$. Conversely, for a row $q \neq 1$, let $\mathrm{pred}(q)$ denote the predecessor reached by following the unique incoming-character edge of $q$. This operation requires select support in the SBWT bit matrix, whose supporting data structures are not stored by default but can be added on demand. In the bit matrix representation, $\mathrm{pred}(q) = \mathrm{subsetselect}_c(q - 1 - C[c])$, where $c$ is the unique character satisfying $1 + C[c] < q \leq 1 + C[c] + |\mathrm{col}_c|$, and $\mathrm{subsetselect}_c(r)$ returns the row containing the $r$-th set bit in column $\mathrm{col}_c$.

# 3 Set operations on SBWTs

## 3.1 Merge plan

Let $A^+$ and $B^+$ be two sets of $k$-mers (including the necessary dummy $k$-mers in their respective SBWTs). In order to support set operations, we first compute a *merge plan* of $A^+$ and $B^+$. Conceptually, this merge plan is a string $T$ of length $|A^+ \cup B^+|$, from the alphabet {L,R,B}, where the symbols stand for L = "left", R = "right" and B = "both". The string is defined as follows:

- $T[i]$ = L if the colexicographically $i$-th $k$-mer in $A^+ \cup B^+$ is found only in $A^+$
- $T[i]$ = R if the colexicographically $i$-th $k$-mer in $A^+ \cup B^+$ is found only in $B^+$
- $T[i]$ = B otherwise (found in both $A^+$ and $B^+$).

With $T$ and the colexicographically sorted lists of $k$-mers in $A^+$ and $B^+$, we can list the $k$-mers of $A^+ \cup B^+$ in colexicographic order by placing a pointer to the starts of $A^+$ and $B^+$, and picking $k$-mers from $A^+$ and $B^+$ according to $T$ while advancing the pointers. This amounts to computing the union of $A^+$ and $B^+$.We can also perform an intersection by reporting only the $k$-mers that are associated with the letter B in $T$, or a set difference by reporting only $k$-mers associated with the letter L for $A^+ \setminus B^+$, or R for $B^+ \setminus A^+$.

### 3.1.1 Computing the merge plan in $O(nk)$ time and $O(n \log \sigma)$ bits

Let $A^+_\ell = \{x[k-\ell+1..k] \mid x \in A^+\}$ be the suffixes of length $\ell$ in $A^+$, and similarly let $B^+_\ell$ be the suffixes of length $\ell$ in $B^+$. Let $P_\ell$ be the *interim merge plan* for $A^+_\ell$ and $B^+_\ell$. We represent $P_\ell$ as a sequence of pairs $(a_0, b_0), \ldots (a_{m-1}, b_{m-1})$, where $m = |A^+_\ell \cup B^+_\ell|$. The number $a_i$ is the number of $k$-mers in $A^+$ that are suffixed by the $i$-th $\ell$-mer of $A^+_\ell \cup B^+_\ell$, and the number $b_i$ is the number of $k$-mers in $B^+$ that are suffixed by the $i$-th $\ell$-mer of $A^+_\ell \cup B^+_\ell$. Some of these numbers can be zero, and we always have $\sum_{i=0}^{m-1} a_i = |A^+|$ and $\sum_{i=0}^{m-1} b_i = |B^+|$.

We initialize the algorithm by setting $P_0 = (|A^+|, |B^+|)$, which corresponds to the 0-length suffix $\epsilon$ shared by all $k$-mers. We then run $k$ rounds to refine $P_0$ incrementally up to $P_k$ as follows. Let $M_X$ be the *$k$-mer matrix* of a set of $k$-mers $X$ with the rows sorted in colexicographic order. Given $P_\ell$, $M_{A^+}$ and $M_{B^+}$, we construct $P_{\ell+1}$ by streaming over pairs $(a_i, b_i)$ in $P_\ell$ while maintaining row pointers for $M_{A^+}$ and $M_{B^+}$. For each pair, we examine the next $a_i$ characters of $M_{A^+}$ and $b_i$ characters of $M_{B^+}$ in the $(\ell+1)$-th column from the right. Because the matrices are sorted colexicographically, these character segments are also internally sorted, letting us partition $(a_i, b_i)$ into up to $|\Sigma| + 1$ sub-pairs via a synchronized linear scan. After $k$ rounds, the only pairs remaining in $P_k$ are $(1, 0)$, $(0, 1)$ and $(1, 1)$, corresponding to letters L, R and B in the merge plan $T$.

To save space, we encode the numbers $a_i$ and $b_i$, in unary such that integer $x \geq 0$ is encoded as $\texttt{rep}(x) = \texttt{0}^x\texttt{1}$. A list of pairs $(a_0, b_0), (a_1, b_1) \ldots$ is then encoded by concatenating the codes $\texttt{rep}(a_0), \texttt{rep}(a_1) \ldots$, and concatenating the codes $\texttt{rep}(b_0), \texttt{rep}(b_1) \ldots$. This takes a total of $O(|A^+ \cup B^+|)$ bits of space. With this, we give up the ability for random access in the list of pairs, but we are still able to stream the pairs in order, which is all we need.

The $k$-mer matrix $M_X$ is never materialized in memory in full – instead, we materialize the matrix column by column from right to left using the SBWT. Algorithm 1 shows how the SBWT is used to transform one column of the $k$-mer matrix into the column on the left. For the first iteration, `column_in` is initialized to the last column of the $k$-mer matrix, which is obtained directly from the $C$-array. Similar algorithms can be found in the literature in the

**Algorithm 1** Transforming a column in the SBWT $k$-mer matrix to the column on the left. In our implementation, the 1-bits in each bit matrix column are enumerated using leading-zero machine instructions to find the next set bit.

**Input:** The SBWT and the associated $C$-array. `column_in`$[0 \ldots n)$: input characters, one per SBWT $k$-mer in colex order.
**Output:** `column_out`$[0 \ldots n)$: output characters

```
1: column_out[0] ← $
2: Pos ← array of length |Σ|
3: for each c ∈ Σ do                          ▷ Initialize output pointers
4:     Pos[c] ← 1 + C[c]
5: for i in [0, n) do
6:     for c ∈ SBWT[i] do
7:         column_out[Pos[c]] ← column_in[i]
8:         Pos[c] ← Pos[c] + 1
```

context of the BOSS data structure [17] and Wheeler graphs [1]. The memory access pattern of the algorithm is $|\Sigma|$ linearly advancing pointers, meaning that most of the computation is confined within the CPU cache, making the algorithm very fast in practice.

The algorithm is also cleanly parallelizable into independent threads by assigning disjoint input and output memory regions for each thread. The work can be split evenly for $t$ output threads by splitting the input array `column_in` into $t$ equal pieces, and using binary search on rank queries to find the ranges in `column_out` corresponding to each input piece.

We implement both a low-memory option that encodes the labels `column_in` and `column_out` in $\lceil \log_2(|\Sigma| + 1) \rceil$ bits per character (+1 for the special dollar symbol), and a faster high-memory option that uses the smallest possible native machine integer types to encode characters, using 8, 16, 32 or 64 bits per character.

### 3.1.2 Computing the merge plan in $O(n \log k)$ time and $O(n \log n)$ bits

We now describe an asymptotically faster approach for computing the merge plan. We do not implement this algorithm in this paper, but include it anyway for completeness.

The algorithm works by applying the technique of prefix doubling to the SBWT graph. We denote the nodes and edges of the first input SBWT graph by $(V_1, E_1)$, and those of the second input SBWT graph by $(V_2, E_2)$. Both graphs are preprocessed so that the source node has a self-loop with the label \$ that is smaller than all other characters. For simplicity, we assume that $k = 2^b$ for some positive integer $b$, so the algorithm runs for $b$ rounds (removing this assumption is trivial in practice). At round $r$, we have a mapping $\pi_r : V_1 \cup V_2 \to \mathbb{Z}$ sorting the nodes according to the incoming $2^r$-mers, that is, $\pi_r(u) = \pi_r(v)$ if the incoming $2^r$-mers to $u$ and $v$ are the same, and otherwise $\pi_r(u) < \pi_r(v)$ if the incoming $2^r$-mer to $u$ is colexicographically smaller than the incoming $2^r$-mer to $v$. After the last round, the merge plan can be read from $\pi_b$.

We initialize $\pi_0$ so that $\pi_0(v)$ is the incoming character to $v$. Now, suppose we have constructed $\pi_r$ and want to construct $\pi_{r+1}$. Since every node in both graphs has in-degree exactly 1, it is always possible to walk backward $2^r$ edges unambiguously. Let $\mathsf{back}(v, 2^r)$ denote the node reached by walking $2^r$ edges backward from $v$. We assign each node $v$ to the pair $(\pi_r(\mathsf{back}(v, 2^r)), \pi_r(v))$. These pairs are compared by using the second element as the primary key, and the first element as the secondary key. With this comparison, the order of the pairs matches the colexicographic order of the incoming $2^{r+1}$-mers. The mapping $\pi_{r+1}$ is obtained by radix-sorting the pairs (right-to-left), and replacing the pairs with single integers preserving the order of the pairs.

To ensure that each round takes linear time, we do not actually walk backward $2^r$ steps from each node, but instead we maintain pointers from $v$ to $\mathsf{back}(v, 2^r)$ at round $r$. The back-pointers for the next round can be constructed by traversing two back-pointers from the current round.

## 3.2 From merge plans to functional SBWT indexes

Given the merge plan, constructing the bit matrix SBWT representation resulting from a set operation on two padded $k$-spectra $A^+$ and $B^+$ is conceptually straightforward. For a union, we keep all positions marked L, R, or B in the merge plan: rows marked L or R are copied from the corresponding input, while rows marked B are obtained by taking the bitwise OR of the two corresponding input rows. For an intersection, we keep only the positions marked B and take the bitwise AND of the two corresponding input rows. For a set difference $A^+ \setminus B^+$, we keep only the positions marked L.

The difficulty is that the SBWT does not encode only the real $k$-mers: it also contains the dummy $k$-mers introduced in the padded $k$-spectra. These dummies represent the minimal set of $k$-mers that must be introduced to guarantee that every non-dummy $k$-mer has an incoming path of length at least $k$ in the SBWT graph, starting at the source node representing $k$-mer $\$^k$. When two SBWTs are combined, some correction is required to keep the dummy $k$-mer set both functional and minimal.

### 3.2.1 Pruning redundant dummy nodes during the union operation

For the union operation, the dummy problem can be described as follows. Let $A^+ = S_k^+(A)$ and $B^+ = S_k^+(B)$ denote the padded $k$-spectra of the two input $k$-mer sets. A naive merge based on the merge plan yields the colexicographically sorted set $A^+ \cup B^+$. However, the desired output is not this set, but rather the padded $k$-spectrum of the union, $S_k^+(A \cup B)$. In the latter, the dummy $k$-mer set is minimal, while the former may contain redundant dummy $k$-mers. When the dummy $k$-mer set is not minimal, the SBWT graph requirement that each node has exactly one incoming-character edge is violated.

**Example.** Consider the case $k = 5$ and the two singleton sets $A = \{\mathsf{ACTGA}\}$ and $B = \{\mathsf{CTGAC}\}$. Their padded $k$-spectra are $A^+ = S_5^+(A) = \{\$\$\$\$\$, \$\$\$\$\mathsf{A}, \$\$\$\mathsf{AC}, \$\$\mathsf{ACT}, \$\mathsf{ACTG}, \mathsf{ACTGA}\}$ and $B^+ = S_5^+(B) = \{\$\$\$\$\$, \$\$\$\$\mathsf{C}, \$\$\$\mathsf{CT}, \$\$\mathsf{CTG}, \$\mathsf{CTGA}, \mathsf{CTGAC}\}$. As shown in Figure 1, the naive merge $A^+ \cup B^+$ is not the canonical padded spectrum of the union $A \cup B$. In the combined set, $\mathsf{ACTGA}$ already provides a predecessor for $\mathsf{CTGAC}$, so dummy $\$\mathsf{CTGA}$ is redundant. The redundancy then propagates backwards along the dummy chain, rendering $\$\$\mathsf{CTG}$, $\$\$\$\mathsf{CT}$, and $\$\$\$\$\mathsf{C}$ redundant as well. Thus, $S_5^+(A \cup B) = \{\$\$\$\$\$, \$\$\$\$\mathsf{A}, \$\$\$\mathsf{AC}, \$\$\mathsf{ACT}, \$\mathsf{ACTG}, \mathsf{ACTGA}, \mathsf{CTGAC}\}$.

**Pruning approach.** Having obtained the merge plan for $A^+$ and $B^+$ and the SBWT bit matrix corresponding to the naive union $A^+ \cup B^+$, redundant dummy nodes can be pruned in three stages (accompanying pseudocode is shown in Algorithm A1).

*Stage 1: mark directly redundant dummies.* We scan each suffix group in $A^+ \cup B^+$, i.e., each maximal interval of consecutive rows whose $k$-mers share the same suffix of length $k-1$.[1] Within a group, a dummy $k$-mer originating exclusively from $A^+$ is *directly redundant* if the same group contains at least one real $k$-mer that is exclusive to $B^+$, or vice versa. For example, in Figure 1, rows 3 and 4 in the merged order form one suffix group, with dummy

[1] These groups are marked in an additional bit vector during merge plan construction.

$A^+ = S_5^+(A)$

| row | $k$-mer | SBWT |
|---|---|---|
| 1 | $$$$$ | A |
| 2 | $$$$A | C |
| 3 | ACTGA | |
| 4 | $$$AC | T |
| 5 | $ACTG | A |
| 6 | $$ACT | G |

$B^+ = S_5^+(B)$

| row | $k$-mer | SBWT |
|---|---|---|
| 1 | $$$$$ | C |
| 2 | $CTGA | C |
| 3 | $$$$C | T |
| 4 | CTGAC | |
| 5 | $$CTG | A |
| 6 | $$$CT | G |

Naive merge $A^+ \cup B^+$

| row | src | $k$-mer | SBWT |
|---|---|---|---|
| 1 | $AB$ | $$$$$ | {A,C} |
| 2 | $A$ | $$$$A | C |
| 3 | $B$ | $CTGA | C |
| 4 | $A$ | ACTGA | |
| 5 | $B$ | $$$$C | T |
| 6 | $A$ | $$$AC | T |
| 7 | $B$ | CTGAC | |
| 8 | $B$ | $$CTG | A |
| 9 | $A$ | $ACTG | A |
| 10 | $B$ | $$$CT | G |
| 11 | $A$ | $$ACT | G |

SBWT graph of $A^+ \cup B^+$

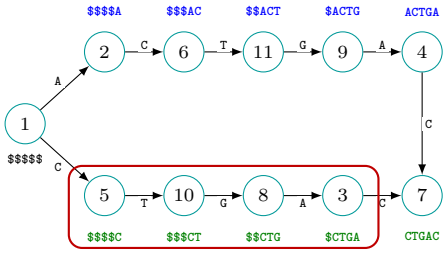



**Figure 1** Naive SBWT merge can create redundant dummy nodes. Example with $k = 5$, $A = \{\mathsf{ACTGA}\}$, and $B = \{\mathsf{CTGAC}\}$. Left: the two padded spectra in colexicographic order, with corresponding SBWT subset sequence. Middle: the naive merged order and subset sequence based on the merge plan. SBWT columns, equivalently the outgoing edge labels in the SBWT graph, are encoded in an SBWT bit matrix; the $k$-mer strings are shown for clarity but not stored. Right: the SBWT graph induced by $A^+ \cup B^+$. The lower dummy chain is redundant because ACTGA already provides a predecessor for CTGAC, so $CTGA and then recursively its dummy predecessors can be pruned.

row 3 exclusive to $B^+$ and real row 4 exclusive to $A^+$. In this case, the real $k$-mer already provides the predecessor that the dummy $k$-mer was introduced to supply. Let $R$ be the set of redundant rows identified in this stage.

*Stage 2: propagate redundancy backwards.* Predecessor nodes in the dummy chains preceding the $k$-mers corresponding to rows in $R$ may also become redundant. We therefore propagate redundancy backwards through the SBWT graph. A dummy row $p = \text{pred}(q)$, for some $q \in R$, is added to $R$ if $p \notin R$ and $\text{succ}(p, c) \in R$ for every $c \in \text{SBWT}[p]$. That is, all successors of $p$ are already redundant, so $p$ carries no useful information. For example, in Figure 1, once row 3 is in $R$, its predecessor row 8 is added because its only successor is row 3, and this then propagates to rows 10 and 5. The propagation is implemented as a level-synchronous breadth-first search and terminates when a full level adds no new rows.

*Stage 3: filter the rows.* Finally, all rows in $R$ are deleted from the merged SBWT bit matrix representing $A^+ \cup B^+$. While scanning the rows, we process one suffix group at a time. Outgoing edge bits from deleted rows are accumulated, except for edges whose successor row is also in $R$. If the suffix group has a surviving row, the accumulated bits are added to the first surviving row of the group. This preserves the SBWT convention that outgoing labels are stored at the first row of each suffix group. The resulting subset sequence is the SBWT of $S_k^+(A \cup B)$.

### 3.2.2 Adding missing dummy nodes during the intersection operation

Whereas for union the naive merge of $A^+$ and $B^+$ could contain redundant dummy $k$-mers, for intersection the opposite problem occurs: a naive row-wise intersection may miss dummy $k$-mers that are required for a valid SBWT index. In such a case, the character counts in the SBWT subset sequence no longer match the character counts in the last column of the colexicographically sorted matrix. Equivalently, the SBWT graph contains source nodes other than the global source $\$^k$. These extra sources are precisely the rows whose missing

$A^+ = S_5^+(A)$

| row | $k$-mer | SBWT |
|---|---|---|
| 1 | $$$$$ | A |
| 2 | $$$$A | C |
| 3 | ACTGA | C |
| 4 | $$$AC | T |
| 5 | CTGAC | |
| 6 | $ACTG | A |
| 7 | $$ACT | G |

$B^+ = S_5^+(B)$

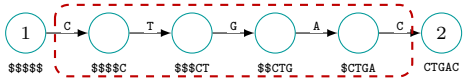
| row | $k$-mer | SBWT |
|---|---|---|
| 1 | $$$$$ | T |
| 2 | TCTGA | C |
| 3 | CTGAC | |
| 4 | $$$TC | T |
| 5 | $TCTG | A |
| 6 | $$$$T | C |
| 7 | $$TCT | G |

Naive intersection $A^+ \cap B^+$

| row | $k$-mer |
|---|---|
| 1 | $$$$$ |
| 2 | CTGAC |

not a valid SBWT index

Desired SBWT graph of $S_5^+(A \cap B)$




missing dummy chain

**Figure 2** Intersection can miss required dummy rows. Example with $k = 5$, $A = \{\mathsf{ACTGA}, \mathsf{CTGAC}\}$, and $B = \{\mathsf{TCTGA}, \mathsf{CTGAC}\}$. Left: the two padded spectra in colexicographic order, with corresponding SBWT subset sequence. Top right: the naive intersection $A^+ \cap B^+$, which keeps only $\$^5$ and $k$-mer CTGAC without any shared outgoing labels, and is not a valid SBWT index. Bottom right: the desired SBWT graph for $S_5^+(A \cap B)$, where the missing dummy chain ending at CTGAC has been added.

dummy predecessors must be added. In other words, intersection cannot be implemented by simply keeping the shared outgoing edges of suffix groups that contain $k$-mers occurring in both input padded spectra. Rather, after identifying shared $k$-mers, we must also add any dummy chains needed to make the resulting set prefix-complete.

**Example.** Consider the case $k = 5$ and the two $k$-mer sets $A = \{\mathsf{ACTGA}, \mathsf{CTGAC}\}$ (from string ACTGAC) and $B = \{\mathsf{TCTGA}, \mathsf{CTGAC}\}$ (from string TCTGAC). Their padded spectra are $A^+ = S_5^+(A) = \{\$\$\$\$\$, \$\$\$\$\mathsf{A}, \$\$\$\mathsf{AC}, \$\$\mathsf{ACT}, \$\mathsf{ACTG}, \mathsf{ACTGA}, \mathsf{CTGAC}\}$ and $B^+ = S_5^+(B) = \{\$\$\$\$\$, \$\$\$\$\mathsf{T}, \$\$\$\mathsf{TC}, \$\$\mathsf{TCT}, \$\mathsf{TCTG}, \mathsf{TCTGA}, \mathsf{CTGAC}\}$. As shown in Figure 2, the naive intersection keeps only the shared $k$-mers $\$\$\$\$\$ and CTGAC, with no shared outgoing labels. This is not a valid SBWT index, because the dummy predecessors $\mathsf{CTGA}, \$\mathsf{CTG}, \$\$\mathsf{CT}$, and $\$\$\$\mathsf{C}$ are missing. The canonical padded spectrum requires adding the dummy chain ending at CTGAC. Thus, $S_5^+(A \cap B) = \{\$\$\$\$\$, \$\$\$\$\mathsf{C}, \$\$\$\mathsf{CT}, \$\$\mathsf{CTG}, \$\mathsf{CTGA}, \mathsf{CTGAC}\}$.

**Missing dummy repair.** Having obtained the merge plan for $A^+$ and $B^+$, we can construct $S_k^+(A \cap B)$ in two major stages. The first stage tests whether the naive intersection is already prefix-complete, where the naive intersection keeps only outgoing labels that appear in both inputs for suffix groups containing rows shared by both $A^+$ and $B^+$. If so, no repair is required. Otherwise, a dummy-repair stage reconstructs the missing predecessor chains by building a new three-way merge plan. Accompanying pseudocode is shown in Algorithm A2.

*Stage 1: incoming-edge coverage.* We scan the suffix groups of the merged order. For each suffix group that contains at least one shared $k$-mer, the naive intersection keeps exactly the outgoing labels that are present in both input SBWTs for that suffix group. For each resulting label $c$, the successor reached by following the corresponding edge in the SBWT graph is marked as having an incoming edge in the intersection. Since the naive intersection may not be a functional SBWT, the successor is computed using rank support in the $A^+$ colexicographic space. After the scan, any intersection $k$-mer that was not marked is a *source node* violating SBWT requirements. If no such new source nodes exist, all non-source intersection rows already have incoming edges and SBWT index construction is trivial. Otherwise, dummy-chain repair is required.[2]

[2] In rare cases, the naive intersection can also retain redundant dummy $k$-mers, prunable using a technique similar to the one described above; we omit the details here for brevity.

*Stage 2a: repair missing dummy chains.* Each new source node is a $k$-mer in the intersection that lacks a predecessor. Let $\mathcal{P}$ contain the colexicographic positions of all new source nodes in the $A^+$ colexicographic space. For each position $p \in \mathcal{P}$, we reconstruct the corresponding $k$-mer by starting at $p$ and repeating $p \leftarrow \text{pred}(p)$ for $k$ steps in $A^+$, recording the incoming edge label at each step. The set $P$ of reconstructed source $k$-mers is then passed to the SBWT builder. This produces a third index representing the padded $k$-spectrum $P^+ = S_k^+(P)$, whose dummy chains supply the missing predecessors.

*Stage 2b: build the repaired intersection.* We compute a three-way merge plan of $A^+$, $B^+$, and $P^+$ in colexicographic order using the $O(nk)$ time implementation. A row contributes to the final intersection if it represents either a shared $k$-mer from $A^+ \cap B^+$ or a row from $P^+$. For each contributing suffix group, outgoing labels are selected by taking those present in $P^+$ or present in both $A^+$ and $B^+$. The resulting subset sequence is the SBWT of $S_k^+(A \cap B)$.

### 3.2.3 Adding missing dummy nodes during the set difference operation

To compute the set difference $A \setminus B$ using the merge plan, we keep rows and corresponding outgoing labels that are present in $A^+$ but not in $B^+$. Constructing the SBWT index in this way needs the same dummy-repair stage as for intersection. In the resulting three-way merge plan, output rows and their labels are selected by taking those present in the padded spectrum $P^+$ or those present in $A^+$ but not in $B^+$. This approach differs from intersection in one additional detail. The global source $\$^k$ is present in both inputs and is thus not selected. It must therefore be inserted explicitly into the naive difference order, with outgoing labels selected as those present in $A^+$ but not in $B^+$ for the global source suffix group.

## 4 Experiments

This section presents experimental results demonstrating the performance of set operations on SBWT indexes. The evaluation has two parts. First, we benchmark SBWT-based union, intersection, and set difference on datasets with different overlap characteristics, reporting runtime, memory usage, and index size alongside two baseline methods: KMC [14] and a straightforward in-house sorted-list implementation. Second, we show that SBWT merging can serve as a practical building block for large-scale index construction by building an SBWT index for the complete 661K bacterial genomes dataset [7] using a binomial merge pipeline.

### 4.1 Benchmarking SBWT set operations

**Data.** We benchmarked set operations on two pairs of datasets chosen to represent different overlap regimes. For a high-overlap setting, we used paternal assembly GCA_018472715.2 and maternal assembly GCA_018472765.3. These correspond to sample HG00735 from the Human Pangenome Reference Consortium [20]. Because human genomes are highly conserved at the sequence level, the $k$-mer spectra of the two haplotype assemblies are expected to overlap strongly. These assemblies were indexed on the forward strand only.

For a lower-overlap setting, we used two diverse sequence collections: the first batch of 1 000 assemblies from the ENA 661K bacterial genomes dataset [7], and the complete PLSDB plasmid database [19]. Because these datasets combine sequences from heterogeneous public sources, both strands were indexed to avoid bias due to assembly or plasmid orientation. Additional dataset statistics are detailed in Supplementary Table A1.

**Environment.** All benchmarks in this subsection were run on a server with an Intel Xeon Gold 6430 CPU, 64 cores, and 503.4 GiB of RAM. Each configuration was run once as a warm-up and then repeated 10 times; reported runtimes are medians over the 10 repetitions.

**Baselines.** We compared SBWT set operations[3] in both default and low-memory modes (Section 3.1.1) to KMC version 3.2.4 and a straightforward in-house sorted-list baseline[4]. All tools were run with $k = 31$. We do not include dynamic $k$-mer set structures [15, 1] in our experiments because they have a different design goal than us (i.e., support for insertion and deletion of individual $k$-mers) and are slow at bulk operations.

KMC is a $k$-mer counting tool that stores distinct $k$-mers, optionally with counts, in a compact database representation. KMC3 builds its database in two stages: first, input $k$-mers are partitioned into bins using short signatures; second, the bins are sorted independently to remove duplicates and compute counts [14]. The set operations then follow a merge-like scan over ordered $k$-mer streams from the two input databases.

The sorted-list baseline encodes each $k$-mer as a 64-bit integer using 2 bits per base, which supports $k \leq 32$. During construction, $k$-mers (and, when requested, their reverse complements) are extracted from the input sequences, sorted with a radix sort, and deduplicated, yielding a sorted array of distinct $k$-mers. Input parsing is performed using the Helicase library [16]. The resulting sets are stored in a compact binary format. Set operations are implemented by a standard two-pointer merge scan over the two sorted arrays, conceptually similar to the approach used by KMC. The scan is performed through buffered iterators over the sorted arrays on disk, so the inputs need not be fully loaded into memory.

We emphasize that KMC and the sorted-list implementation serve as *baselines* for exact $k$-mer set operations. Both represent sets in forms that are well suited to ordered scans and database-style set manipulation. The SBWT has a different goal: it represents the set as a *compressed index* that supports membership queries in time independent of the number of indexed $k$-mers, and navigation between overlapping $k$-mers in the SBWT graph. Thus, the comparison below is aimed at *measuring the overhead of preserving this richer indexed structure* while performing set operations, rather than comparing interchangeable structures.

### 4.1.1 High-overlap benchmark: human haplotypes

Table 1 reports the build and set-operation statistics for the human haplotype benchmark. The set-operation measurements from the table (together with supplementary thread counts) are visualized in Figure 3, which shows the runtime-memory trade-off for union, intersection, and set difference. As reported in Table 1, the two assemblies contain almost the same number of distinct 31-mers. As expected in this high-overlap setting, the intersection is large, the difference is much smaller, and the union remains close in size to the two input sets.

The most striking difference between the three methods is index size. For both input haplotypes, the SBWT indexes are about an order of magnitude smaller than the corresponding KMC and sorted-list representations. The same pattern holds approximately for the set-operation outputs. This reflects one main benefit of the SBWT representation: the output is a *compressed index* of the $k$-mer set, rather than an ordered $k$-mer database or list.

[3] v0.1.0, Rust implementation available at `https://github.com/LoreDepuydt/sbwt-set-operations`
[4] v0.1.0, Rust implementation available at `https://github.com/LoreDepuydt/sorted-lists`

**Table 1** Performance metrics for building and performing set operations on 31-mer indexes of two human haplotypes (HG00735). Values represent the median of 10 repetitions. SBWT and KMC used 64 threads; the sorted-lists baseline was single-threaded. SBWT is slower than the baselines for set operations, but produces compressed indexes that are about an order of magnitude smaller than those of the baseline tools, while preserving a queryable and navigable representation of the $k$-mer set. Detailed command-line settings are given in Supplementary Table A2.

| | Assemblies | | Set Operations | | |
|---|---|---|---|---|---|
| | paternal | maternal | pat. ∪ mat. | pat. ∩ mat. | pat. \ mat. |
| # 31-mers (Millions) | 2549.46 | 2548.23 | 2635.99 | 2461.70 | 87.76 |
| **SBWT** ($t64$) | | | | | |
| Index size (GiB) | **1.48** | **1.48** | **1.54** | **1.46** | **0.08** |
| Runtime (s) | 59.49 | 60.19 | 94.66 | 188.59 | 190.49 |
| *low-memory* | | | 148.74 | 295.66 | 294.80 |
| Peak RAM (GiB) | **27.65** | **27.66** | 16.78 | 21.04 | 21.12 |
| *low-memory* | | | 9.84 | 13.73 | 14.16 |
| **KMC** ($t64$) | | | | | |
| Index size (GiB) | 19.00 | 18.99 | 14.76 | 13.79 | 0.52 |
| Runtime (s) | **36.96** | **36.70** | **26.99** | **26.69** | **18.75** |
| Peak RAM (GiB) | 30.50 | 30.33 | **7.27** | **7.28** | 7.28 |
| **Sorted lists** ($t1$) | | | | | |
| Index size (GiB) | 18.99 | 18.99 | 19.64 | 18.34 | 0.65 |
| Runtime (s) | 194.93 | 196.35 | 58.99 | 58.37 | 25.04 |
| Peak RAM (GiB) | 45.48 | 45.56 | 19.64 | 18.34 | **0.66** |

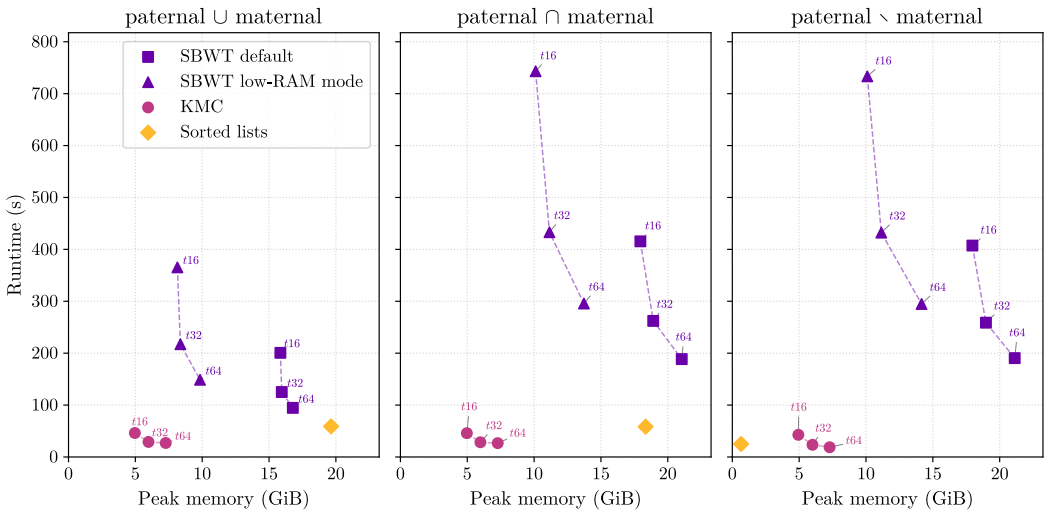



**Figure 3** Runtime vs. peak memory for three 31-mer set operations on the paternal and maternal haplotype assemblies of sample HG00735. Thread counts of 16, 32, and 64 are shown for the multi-threaded tools (KMC, SBWT default, and SBWT low-memory mode). The sorted-lists baseline is single-threaded. The $t64$ values correspond to the set-operation columns of Table 1. The baseline tools (KMC and sorted lists) perform ordered $k$-mer set manipulation, whereas SBWT set operations additionally construct a richer indexed representation supporting $k$-mer lookup and SBWT graph navigation; this extra indexing work explains the higher runtime and memory usage, and yields an order-of-magnitude smaller output index.

When comparing set-operation performance, KMC (using 64 threads) is the fastest method for all three set operations. The sorted-list baseline is somewhat slower but competitive despite being single-threaded. Relative to KMC, default SBWT is 3.5× slower for union, 7.1× slower for intersection, and 10.2× slower for difference. This overhead can be explained by the fact that the SBWT algorithms operate on compactly encoded indexes and must construct a valid SBWT index (including the dummy-node corrections described in Section 3.2), rather than merely reading and writing ordered collections of naively encoded $k$-mers. Profiling on the human benchmark shows that merge-plan construction (Section 3.1.1) dominates SBWT set-operation runtime, accounting for 86.7–93.5% of union time and 93.1–97.2% of intersection and difference time (including the three-way merge plan used for dummy repair).

KMC uses the least RAM during the set operations. The sorted-list baseline has peak memory usage close to its output size, because the scan itself is streaming but the output set is stored explicitly in memory. This explains why sorted lists use little memory for the small-difference output, but substantially more memory for the much larger union and intersection outputs. For SBWT, intersection and difference use more memory than union because these operations require a three-way merge plan construction, whereas union uses only the two-way merge plan. The low-memory SBWT mode (see Section 3.1.1) reduces peak memory significantly relative to the default mode, at the cost of increased runtime.

### 4.1.2 Low-overlap benchmark: bacteria and plasmids

Supplementary Table A4 and Figure A1 show the bacteria–plasmid benchmark. In contrast to the human haplotypes, there is little overlap: the intersection is small, the union is close to the sum of the two inputs, and most bacterial 31-mers are absent from the plasmids.

The same overall pattern is observed as in the high-overlap benchmark: SBWT produces substantially smaller indexes than both baselines, KMC is again the fastest method for all three set operations, and relative to KMC, default SBWT is 2.3× slower for union, 6.0× slower for intersection, and 5.8× slower for difference. Runtime for the sorted-list baseline is close to SBWT for union, but lower for intersection and difference, where the output is smaller. As in the human benchmark, merge-plan construction dominates the SBWT runtime, accounting for 84.2–90.0% of union time and 93.9–97.2% of intersection and difference time.

The memory trends are also consistent with the human benchmark: KMC uses the least RAM during set operations, sorted lists have peak memory close to output size, and SBWT intersection and difference require more memory than union because they use a three-way merge plan construction. As before, the low-memory SBWT mode reduces peak memory relative to the default mode at the cost of increased runtime.

## 4.2 Building the 661K SBWT index through a binomial merge pipeline

**Data.** We evaluate large-scale SBWT construction through a binomial merge pipeline on the curated ENA 661K bacterial genomes dataset [7]. The full input contains 2.57 Tbp of sequence data across 661 405 assemblies. ENA distributes this dataset in 662 batches of 1 000 assemblies.

**Environment.** Most of the binomial merge pipeline was run on a server with two Intel Xeon Gold 6130 CPUs, 32 physical cores, and 128 GiB of RAM. The final merge was run on the server used for the set-operation benchmarks (Intel Xeon Gold 6430, 64 physical cores, 503.4 GiB RAM), because it required more memory. The full pipeline was run once.

**Table 2** Summary of the binomial merge pipeline for construction of the SBWT index of the 661K bacterial genomes dataset (2.57 Tbp). The final root index contains $88.70 \times 10^9$ distinct 31-mers and occupies 51.73 GiB on disk. The pipeline performed 662 batch builds and 661 pairwise merges.

| **Metric** | **Batch builds** ($n = 662$) | **Merges** ($n = 661$) |
|---|---|---|
| Total runtime | 19h 32m 15s | 30h 58m 57s |
| Fraction of total runtime | 38.7% | 61.3% |
| Mean runtime | 1m 46s ± 30s | 2m 48s ± 7m 32s |
| Median runtime | 1m 46s | 1m 03s |
| Min / max runtime | 45s / 4m 16s | 5s / 1h 37m 22s |
| Mean peak RAM | 40.94 GiB | 7.37 GiB |
| Median peak RAM | 41.94 GiB | 3.58 GiB |
| Max peak RAM | 57.36 GiB | **186.23 GiB** |
| **Pipeline total** | | **50h 31m 13s** |

**Binomial merge pipeline design.** The goal of the pipeline is to avoid constructing the full 661K index in one monolithic build, which is expensive in terms of either RAM or temporary disk usage. Instead, we gradually construct SBWT indexes for the ENA batches of 1 000 assemblies and merge these indexes incrementally. This turns large-scale construction into a sequence of smaller batch builds and pairwise merge operations.

We organize the pairwise merges using an online binomial merge stack. Each stack level stores at most one SBWT index: level 0 stores one batch, level 1 stores a merge of two batches, level 2 stores a merge of four batches, and so on. Whenever a newly built or newly merged index reaches an occupied level, the two indexes at that level are merged and carried to the next level. After all batches have been processed, the remaining stack entries are flushed by pairwise merges until a single final index remains.

In the implementation, the pipeline uses $k = 31$ and indexes both strands. For each ENA batch, the corresponding FASTA files are prefetched in the background, an SBWT index is built, and the downloaded FASTA files are deleted after construction. Batch indexes are built with 32 threads using the in-memory SBWT construction algorithm and a RAM usage target of 48 GiB. Merges are run with 32 threads. The low-memory merge mode is used for merge levels at least 5, or as a fallback if a default merge fails. See also Table A3 for command templates.

Table 2 summarizes the full 661K pipeline. The final index contains 88.70 billion distinct 31-mers and occupies 51.73 GiB on disk. The full pipeline completed in 50h 31m 13s, with batch construction accounting for 38.7% of the measured runtime and merges accounting for the remaining 61.3%. Peak memory usage occurred in the final root merge.

### 4.2.1 Comparison to direct SBWT construction

We compared the merge pipeline against direct SBWT construction [6] on prefixes of the 661K dataset. For each prefix, we used the first $2^i$ ENA batches, as far as computing resources allowed. We compared the binomial merge pipeline to both the in-memory SBWT construction algorithm and the disk-assisted construction algorithm. For the binomial merge pipeline, the cumulative runtime for a prefix of $2^i$ batches is computed as the sum of all batch-build and merge times in the corresponding merge subtree. Peak RAM is the maximum resident set size observed over all steps in that subtree. Peak temporary disk usage is the sum of index sizes of intermediate SBWT files that coexist during the binomial merge process.

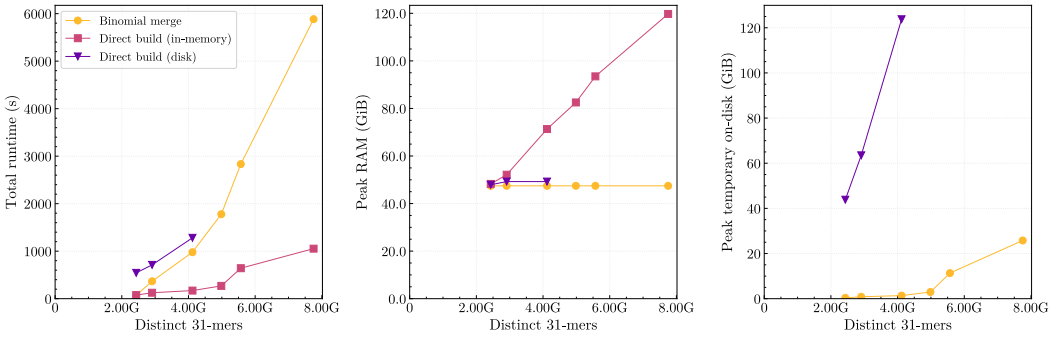



**Figure 4** Scaling behavior of the binomial merge pipeline versus direct SBWT construction [6] as a function of the number of distinct 31-mers in the resulting index, evaluated on prefixes of the 661K dataset consisting of the first $2^k$ ($k = 0, 1, \ldots, 5$) ENA batches of 1 000 bacterial assemblies. The direct-construction curves stop where the corresponding method exceeded the available RAM or temporary-disk limits. Left: total runtime. Centre: peak RAM usage. Right: peak temporary on-disk storage. The binomial merge pipeline is slower than direct in-memory construction on the tested prefixes, but keeps RAM and temporary-disk usage within practical limits.

Figure 4 shows the scaling behavior of the binomial merge pipeline compared with direct construction on the dataset prefixes and demonstrates why the merge pipeline is necessary at this scale when memory is limited. Direct in-memory construction has lower runtime on the prefixes where it fits in memory, but its peak RAM increases rapidly with the number of distinct 31-mers. Conversely, disk-assisted construction reduces RAM pressure, but requires substantially more temporary disk space. Both direct-construction curves stop once the corresponding resource requirement exceeds the RAM or disk-space limits on our machine.

The binomial merge pipeline trades some additional runtime for bounded memory and disk usage. Over the prefixes shown in Figure 4, peak RAM remains close to the per-batch construction limit, and temporary disk usage grows much more slowly than for the disk-assisted direct build. For larger indexes, merge memory eventually becomes the dominant memory cost; this transition is visible in the full merge-tree analysis in Figure 6. A rough linear extrapolation from the direct-build measurements suggests that constructing the full 661K SBWT index directly would require on the order of 1.2 TiB of RAM with the in-memory algorithm, or 5.6 TiB of temporary disk space with the disk-assisted algorithm – prohibitive amounts for most servers. By contrast, the binomial merge pipeline completed the full construction in 50h 31m 13s, with a peak RAM usage of 186.23 GiB and a peak disk usage of 112.09 GiB. This makes the binomial merge pipeline the most *practical* construction strategy for the full dataset on this hardware.

### 4.2.2 Merge-tree scaling and resource distribution

Figure 5 shows how the number of distinct 31-mers grows across all nodes of the binomial merge tree. Each leaf corresponds to one ENA batch of 1 000 assemblies, and each internal node corresponds to a merged SBWT index. As merge depth increases, the points move to the right and upward, reflecting more input sequence and more distinct 31-mers. However, the horizontal growth is stronger than the vertical growth, showing that distinct 31-mers accumulate sublinearly with input sequence length. Thus, as more batches are merged, increasing amounts of input sequence are represented by comparatively fewer new 31-mers. At the root, the final index contains 88.70 billion distinct 31-mers over 2.57 Tbp of input sequence, corresponding to approximately one distinct 31-mer per 29 input bases.

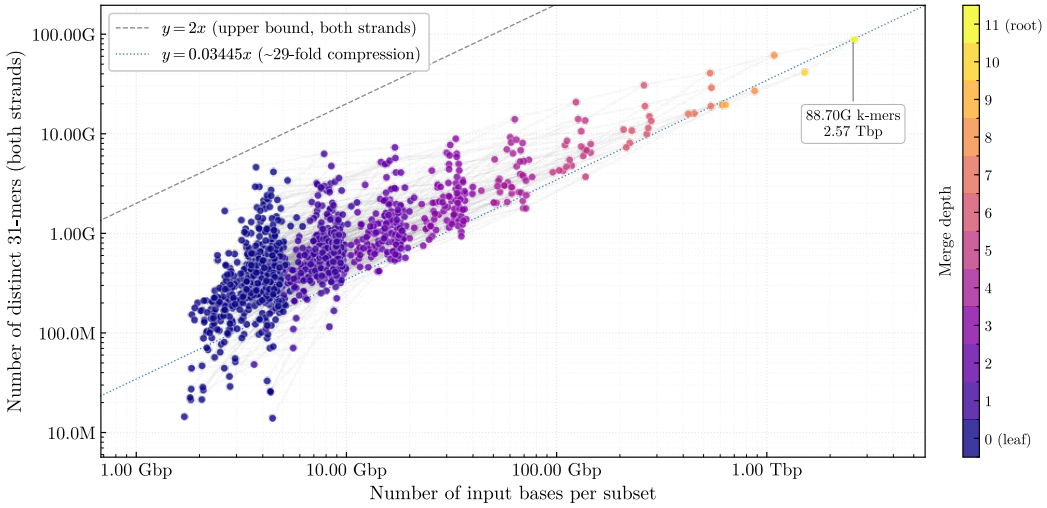



**Figure 5** Scaling analysis of the number of distinct 31-mers as a function of total input sequence length for all nodes in the SBWT binomial merge tree for the 661K dataset. Both axes are logarithmic. Each point represents one node: either a single-batch leaf (1 000 assemblies) or a merged internal node. Color encodes merge depth in the tree. Internal nodes are connected to their two children, tracing the successive pairwise merges of the binomial tree. The dashed line $y = 2x$ is the theoretical upper bound on the number of distinct 31-mers when indexing both strands. The dotted line $y = 0.03445\,x$ shows the empirical rate at which distinct 31-mers accumulate in the full dataset: approximately one distinct 31-mer per 29 input bases. Distinct 31-mers accumulate much more slowly than input sequence length, reflecting substantial redundancy in the dataset.

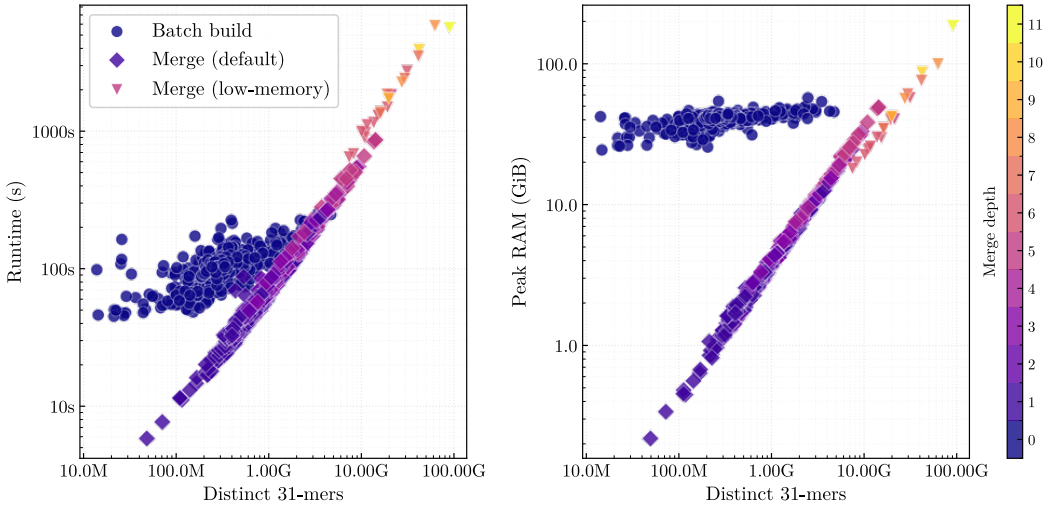



**Figure 6** Runtime and peak memory usage as a function of the number of distinct 31-mers for all nodes in the SBWT binomial merge tree for the 661K dataset. Each point represents one pipeline stage: a single-batch index build or a pairwise merge operation. Color encodes merge depth. Left: runtime in seconds. Right: peak resident set size in GiB. Both axes are logarithmic. Batch builds have nearly fixed peak memory due to the construction memory target (48 GiB), whereas merge time and memory cost increase with $k$-mer set size. Low-memory merges are distinguished because they trade runtime for reduced memory pressure; they appear for the largest $k$-mer counts, where standard merging exceeded predefined RAM limits, or above merge level 5.

Figure 6 shows where time and memory are spent across the merge tree. Batch builds have relatively stable peak memory, because they use a fixed memory target, while their runtimes vary somewhat due to batch size and sequence content. Merge operations show a clear dependence on the number of distinct 31-mers in the indexes being merged: both runtime and memory increase with merge size. Small and medium merges are inexpensive compared with batch construction, while the largest merge levels dominate the tail of the runtime and determine the peak memory requirements of the full pipeline. The low-memory merge mode appears at larger merge sizes and higher merge depths. As expected, low-memory mode shifts points toward higher runtime in exchange for lower memory usage.

## 5 Concluding Remarks

We have presented practical algorithms for the three fundamental set operations, union, intersection, and set difference, on $k$-spectra represented as spectral Burrows-Wheeler transforms. At the core of our approach is a compact $\mathcal{O}(n)$-bit *merge plan*, constructible in $\mathcal{O}(nk)$ time, together with linear-time post-processing stages that repair the dummy padding so that the output is itself a valid, queryable SBWT index. As an immediate consequence, we have described a more scalable SBWT construction algorithm.

There is much room for future work. On a technical front, our most pressing follow-up experiment is to gauge the effect of the batch size at the leaves of the merge tree on overall runtime performance. It would also be interesting to see if the $O(n \log k)$ algorithm we have described translates into a practical improvement in set operations, especially because merge-plan construction accounts for the largest fraction of SBWT set-operation runtime in our experiments. Also, while the 661K dataset is already of non-trivial size, in the full version of this paper we plan to include experiments with even larger datasets, such as AllTheBacteria [12]. The merge plan labels could also be repurposed for indexing $k$-mers with provenance information.

Probably the most interesting theoretical problem we leave open is the existence, or non-existence, of an $O(n)$-time $k$-mer set union algorithm. Recent work by Alanko and Pérez-López [5], which considers the tangentially related problem of counting $k$-mers in deterministic Wheeler graphs, suggests this may be difficult. The two counting algorithms they describe take $O(nk)$ or $O(n^4 \log k)$ time. Applying these counting algorithms to a union would require determinizing the combined Wheeler graph of two $k$-mer sets, with potentially exponential blow-up; whether the SBWT avoids this is open.

## References


1 Jarno Alanko, Bahar Alipanahi, Jonathen Settle, Christina Boucher, and Travis Gagie. Buffering updates enables efficient dynamic de Bruijn graphs. *Computational and Structural Biotechnology Journal*, 19:4067–4078, January 2021. doi:10.1016/j.csbj.2021.06.047.

2 Jarno N. Alanko, Elena Biagi, Joel Mackenzie, and Simon J. Puglisi. Batched k-Mer Lookup on the Spectral Burrows-Wheeler Transform. In Rezaul Chowdhury, Jonathan W. Berry, Kathrin Hanauer, and Bin Ren, editors, *Proceedings of the 27th Symposium on Algorithm Engineering and Experiments, ALENEX 2025, New Orleans, LA, USA, January 12–13, 2025*, pages 95–106. SIAM, January 2025. doi:10.1137/1.9781611978339.8.

3 Jarno N. Alanko, Elena Biagi, and Simon J. Puglisi. Longest Common Prefix Arrays for Succinct k-Spectra. In Franco Maria Nardini, Nadia Pisanti, and Rossano Venturini, editors, *String Processing and Information Retrieval - 30th International Symposium, SPIRE 2023, Pisa, Italy, September 26–28, 2023, Proceedings*, volume 14240 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2023. doi:10.1007/978-3-031-43980-3_1.

4 Jarno N. Alanko, Lore Depuydt, Camille Marchet, and Simon J. Puglisi. LoreDepuydt/sbwt-set-operations. Software, swhId: `swh:1:dir:de3c949dbd910f97f6af25f8be4838be2fbe4dfb` (visited on 2026-08-14). URL: `https://github.com/LoreDepuydt/sbwt-set-operations`, `doi:10.4230/artifacts.27674`.

5 Jarno N. Alanko and Máximo Pérez-López. Computing k-mers in Graphs. In Philip Bille and Nicola Prezza, editors, *37th Annual Symposium on Combinatorial Pattern Matching, CPM 2026, Copenhagen, Denmark, June 15–17, 2026*, volume 369 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 3:1–3:17, Dagstuhl, Germany, 2026. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.CPM.2026.3`.

6 Jarno N. Alanko, Simon J. Puglisi, and Jaakko Vuohtoniemi. Small Searchable $\kappa$-Spectra via Subset Rank Queries on the Spectral Burrows-Wheeler Transform. In Jonathan W. Berry, David B. Shmoys, Lenore Cowen, and Uwe Naumann, editors, *SIAM Conference on Applied and Computational Discrete Algorithms, ACDA 2023, Seattle, WA, USA, May 31–June 2, 2023*, pages 225–236. SIAM, January 2023. `doi:10.1137/1.9781611977714.20`.

7 Grace A. Blackwell, Martin Hunt, Kerri M. Malone, Leandro Lima, Gal Horesh, Blaise T. F. Alako, Nicholas R. Thomson, and Zamin Iqbal. Exploring bacterial diversity via a curated and searchable snapshot of archived DNA sequences. *PLOS Biology*, 19(11):1–16, November 2021. `doi:10.1371/journal.pbio.3001421`.

8 Rayan Chikhi, Brice Raffestin, Anton Korobeynikov, Robert Edgar, and Artem Babaian. Logan: Planetary-Scale Genome Assembly Surveys Life's Diversity. *bioRxiv*, 2024. `doi:10.1101/2024.07.30.605881`.

9 Bastien Degardins, Charles Paperman, and Camille Marchet. Vizitig: a pangenome and pantranscriptome explorer. *bioRxiv*, 2026. `doi:10.1101/2025.04.19.649656`.

10 Lavinia Egidi, Felipe A. Louza, and Giovanni Manzini. Space Efficient Merging of de Bruijn Graphs and Wheeler Graphs. *Algorithmica*, 84(3):639–669, 2022. `doi:10.1007/s00453-021-00855-2`.

11 Guillaume Holley and Páll Melsted. Bifrost: highly parallel construction and indexing of colored and compacted de Bruijn graphs. *Genome Biology*, 21:249, September 2020. `doi:10.1186/s13059-020-02135-8`.

12 Martin Hunt, Leandro Lima, Daniel Anderson, George Bouras, Michael Hall, Jane Hawkey, Oliver Schwengers, Wei Shen, John A. Lees, and Zamin Iqbal. AllTheBacteria – all bacterial genomes assembled, available, and searchable. *bioRxiv*, 2025. `doi:10.1101/2024.03.08.584059`.

13 Mikhail Karasikov, Harun Mustafa, Daniel Danciu, Oleksandr Kulkov, Marc Zimmermann, Christopher Barber, Gunnar Rätsch, and André Kahles. Efficient and accurate search in petabase-scale sequence repositories. *Nature*, 647(8091):1036–1044, October 2025. `doi:10.1038/s41586-025-09603-w`.

14 Marek Kokot, Maciej Długosz, and Sebastian Deorowicz. KMC 3: counting and manipulating k-mer statistics. *Bioinformatics*, 33(17):2759–2761, September 2017. `doi:10.1093/bioinformatics/btx304`.

15 Igor Martayan, Bastien Cazaux, Antoine Limasset, and Camille Marchet. Conway-Bromage-Lyndon (CBL): an exact, dynamic representation of $k$-mer sets. *Bioinformatics*, 40(Supplement_1):i48–i57, July 2024. `doi:10.1093/bioinformatics/btae217`.

16 Igor Martayan, Loup Lobet, Camille Marchet, and Charles Paperman. Helicase: Vectorized parsing and bitpacking of genomic sequences. *bioRxiv*, 2026. Accepted to RECOMB-Arch 2026. `doi:10.64898/2026.03.19.712912`.

17 Martin D. Muggli, Bahar Alipanahi, and Christina Boucher. Building large updatable colored de Bruijn graphs via merging. *Bioinformatics*, 35(14):i51–i60, 2019. `doi:10.1093/bioinformatics/btz350`.

18 Giulio Ermanno Pibiri. Sparse and skew hashing of K-mers. *Bioinformatics*, 38(Supplement_1):i185–i194, June 2022. `doi:10.1093/bioinformatics/btac245`.

19 Georges Pierre Schmartz, Anna Hartung, Pascal Hirsch, Fabian Kern, Tobias Fehlmann, Rolf Müller, and Andreas Keller. PLSDB: advancing a comprehensive database of bacterial plasmids. *Nucleic Acids Research*, 50(D1):D273–D278, November 2021. doi:10.1093/nar/gkab1111.

20 The Human Pangenome Reference Consortium. A draft human pangenome reference. *Nature*, 617(7960):312–324, May 2023. doi:10.1038/s41586-023-05896-x.

# A Appendix

**Algorithm A1** Pruning redundant dummy rows from the naively merged SBWT of $A^+ \cup B^+$.

**Input:** Naively merged SBWT for $A^+ \cup B^+$, merge plan, suffix-group boundaries bit vector
**Output:** SBWT of $S_k^+(A \cup B)$

$R \leftarrow \emptyset$
**for** each suffix group $G$ **do** ▷ *Stage 1, using suffix-group boundaries bit vector*
  *hasRealA* $\leftarrow \exists$ non-dummy row in $G$ exclusive to $A^+$
  *hasRealB* $\leftarrow \exists$ non-dummy row in $G$ exclusive to $B^+$
  **if** *hasRealB* **then**
    add all dummy rows exclusive to $A^+$ in $G$ to $R$
  **if** *hasRealA* **then**
    add all dummy rows exclusive to $B^+$ in $G$ to $R$
*frontier* $\leftarrow R$
**while** *frontier* $\neq \emptyset$ **do** ▷ *Stage 2*
  *next* $\leftarrow \emptyset$
  **for** each $q \in$ *frontier* **do** ▷ *parallel*
    **if** $q \neq 1$ **then**
      $p \leftarrow \text{pred}(q)$
      **if** $p$ is a dummy row, $p \notin R$, and $\text{succ}(p, c) \in R$ for every $c \in \text{SBWT}[p]$ **then**
        add $p$ to *next*
  $R \leftarrow R \cup$ *next*
  *frontier* $\leftarrow$ *next*
**for** each suffix group $G$ **do** ▷ *Stage 3, using suffix-group boundaries bit vector*
  *bits* $\leftarrow \emptyset$
  *firstSurviving* $\leftarrow \perp$
  **for** each row $p \in G$ **do**
    **if** $p \notin R$ and *firstSurviving* $= \perp$ **then**
      *firstSurviving* $\leftarrow p$
    **if** $p \in R$ **then**
      **for** each $c \in \text{SBWT}[p]$ **do**
        **if** $\text{succ}(p, c) \notin R$ **then**
          set $c$ in *bits*
  **if** *firstSurviving* $\neq \perp$ **then**
    $\text{SBWT}[\textit{firstSurviving}] \leftarrow \text{SBWT}[\textit{firstSurviving}] \cup \textit{bits}$
Delete all rows in $R$
**return** pruned SBWT

**Algorithm A2** Constructing the SBWT intersection from the merge plan of $A^+$ and $B^+$.

**Input:** $A^+$, $B^+$, merge plan, suffix-group boundaries bit vector
**Output:** SBWT of $S_k^+(A \cap B)$

*covered* $\leftarrow \{1\}$ ▷ *global source does not require an incoming edge*
**for** each suffix group $G$ containing a shared row **do** ▷ *Stage 1*
    $L_A \leftarrow$ outgoing labels of $G$ contributed by $A^+$
    $L_B \leftarrow$ outgoing labels of $G$ contributed by $B^+$
    $p_A \leftarrow$ the first row of $A^+$ corresponding to suffix group $G$
    **for** each $c \in L_A \cap L_B$ **do**
        add $\text{succ}_{A^+}(p_A, c)$ to *covered*
$\mathcal{P} \leftarrow \emptyset$
**for** each shared row $p$ **do**
    **if** the $A^+$ colex position $p_A$ of $p$ is not in *covered* **then**
        add $p_A$ to $\mathcal{P}$
**if** $\mathcal{P} = \emptyset$ **then**
    **for** each suffix group $G$ containing a shared row **do**
        $L_A \leftarrow$ outgoing labels of $G$ contributed by $A^+$
        $L_B \leftarrow$ outgoing labels of $G$ contributed by $B^+$
        write $L_A \cap L_B$ at the first shared row of $G$
    **return** assembled intersection SBWT
$P \leftarrow \emptyset$
**for** each $p \in \mathcal{P}$ **do** ▷ *Stage 2a*
    $x \leftarrow p$
    $s \leftarrow \epsilon$
    **for** $i \leftarrow 1$ to $k$ **do**
        let $c$ be the incoming edge label of $x$ in $A^+$
        $s \leftarrow c\,s$
        $x \leftarrow \text{pred}_{A^+}(x)$
    add $s$ to $P$
build $P^+ = S_k^+(P)$ ▷ *using SBWT build implementation*
compute a three-way merge plan of $A^+$, $B^+$, and $P^+$
**for** each suffix group $G$ in the three-way merge plan **do** ▷ *Stage 2b*
    $L_A \leftarrow$ outgoing labels of $G$ contributed by $A^+$
    $L_B \leftarrow$ outgoing labels of $G$ contributed by $B^+$
    $L_P \leftarrow$ outgoing labels of $G$ contributed by $P^+$
    **if** $G$ contains a shared row or a row from $P^+$ **then**
        write $L_P \cup (L_A \cap L_B)$ at the first result row of $G$
**return** corrected SBWT

**Table A1** Input datasets used for the set-operation benchmarks. The table reports the number of input assemblies or records, total sequence length, and gzip-compressed FASTA size.

| Dataset | Assemblies/records | Total length | Compressed size |
|---|---|---|---|
| HG00735 maternal haplotype | 1 | 3.03 Gbp | 860 MiB |
| HG00735 paternal haplotype | 1 | 3.03 Gbp | 861 MiB |
| Plasmids | 72 556 | 7.30 Gbp | 2.1 GiB |
| Bacterial assemblies | 1 000 | 4.21 Gbp | 1.3 GiB |

**Table A2** Command-line settings used for the set-operation benchmarks.

| Tool | Stage | Datasets | Settings |
|---|---|---|---|
| Sorted lists | build | human haplotypes | `sorted_lists build -k 31` |
| Sorted lists | build | plasmids, bacteria | `sorted_lists build -k 31 --strand both` |
| Sorted lists | set ops | all | `sorted_lists merge`, `intersect`, `difference` |
| KMC | build | human haplotypes | `kmc -k31 -m32 -t64 -ci1 -fm -b -r` |
| KMC | build | plasmids, bacteria | same as above, on preprocessed input including reverse complements |
| KMC tools | union | all | `kmc_tools -t<T> simple union -ocsum` |
| KMC tools | intersection | all | `kmc_tools -t<T> simple intersect -ocmin` |
| KMC tools | difference | all | `kmc_tools -t<T> simple kmers_subtract` |
| SBWT | build | human haplotypes | `sbwt build -k 31 -t 64 --in-memory` |
| SBWT | build | plasmids, bacteria | `sbwt build -k 31 -t 64 --mem-gb 32 --add-revcomp --dedup-batches --in-memory` |
| SBWT | set ops | all | `merge`, `intersect`, `difference` `-t<T>` [`--low-ram`] |

**Table A3** Command templates used to construct the 661K SBWT index through a binomial merge pipeline. The `--low-ram` flag is used for merge levels at least 5, or as a fallback when a default merge fails.

| Step | Command template |
|---|---|
| Batch build, binomial pipeline | `sbwt build LIST -k 31 --mem-gb 48 --threads 32 --add-revcomp --dedup-batches --in-memory` |
| Merge, binomial pipeline | `sbwt merge --threads 32` [`--low-ram`] |
| Direct build, in-memory | `sbwt build LIST -k 31 --mem-gb 48 --threads 32 --add-revcomp --dedup-batches --in-memory` |
| Direct build, disk-backed | `sbwt build LIST -k 31 --mem-gb 48 --threads 32 --add-revcomp --dedup-batches --temp-dir TMP` |

**Table A4** Performance metrics for building and performing set operations on 31-mer indexes for 1 000 bacterial assemblies and a plasmid dataset. Values represent the median of 10 repetitions. SBWT and KMC used 64 threads; the sorted-lists baseline was single-threaded. SBWT is slower than the baselines for set operations, but produces compressed indexes that are about an order of magnitude smaller than those of the baseline tools, while preserving a queryable and navigable representation of the $k$-mer set. Detailed command-line settings are given in Supplementary Table A2.

| | Datasets | | Set Operations | | |
|---|---|---|---|---|---|
| | bacteria | plasmids | bac. ∪ pla. | bac. ∩ pla. | bac. \ pla. |
| # 31-mers (Millions) | 2432.09 | 3947.86 | 6225.33 | 154.61 | 2277.47 |
| **SBWT** ($t64$) | | | | | |
| Index size (GiB) | **1.42** | **2.30** | **3.63** | **0.11** | **1.36** |
| Runtime (s) | 76.76 | 127.00 | 161.53 | 307.76 | 316.34 |
| *low-memory* | | | 235.51 | 447.71 | 450.37 |
| Peak RAM (GiB) | 47.74 | 60.08 | 21.71 | 27.40 | 27.98 |
| *low-memory* | | | 15.32 | 17.76 | 18.20 |
| **KMC** ($t64$) | | | | | |
| Index size (GiB) | 18.12 | 29.42 | 34.82 | 0.90 | 12.76 |
| Runtime (s) | **67.04** | **110.44** | **71.03** | **51.35** | **54.36** |
| Peak RAM (GiB) | **34.99** | **40.40** | **7.25** | 7.18 | **7.18** |
| **Sorted lists** ($t1$) | | | | | |
| Index size (GiB) | 18.12 | 29.41 | 46.38 | 1.15 | 16.97 |
| Runtime (s) | 640.70 | 1119.98 | 150.58 | 58.10 | 85.99 |
| Peak RAM (GiB) | 125.42 | 217.44 | 46.38 | **1.15** | 16.97 |

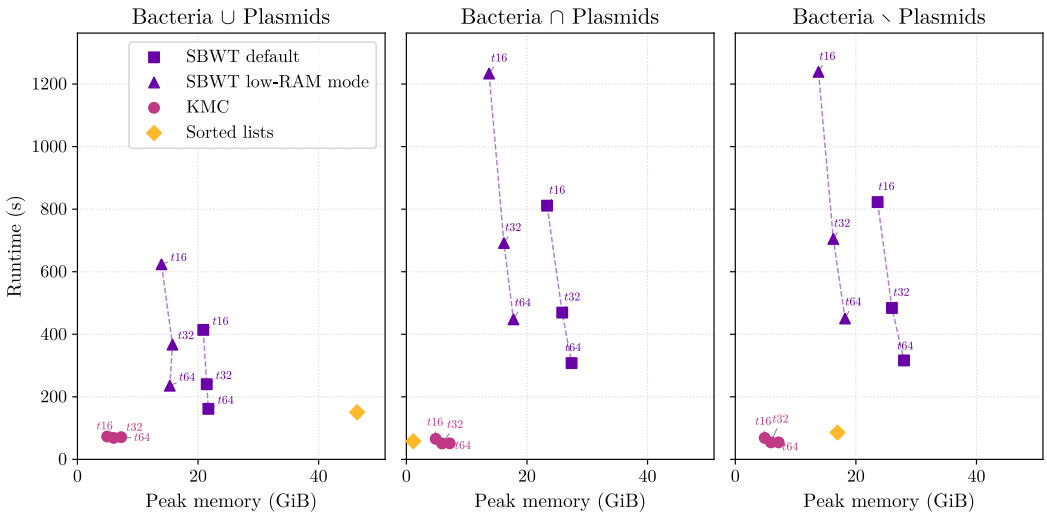



**Figure A1** Runtime vs. peak memory for three 31-mer set operations on 1 000 bacterial assemblies and a plasmid dataset. Each point reports the median over 10 timed repetitions; thread counts of 16, 32, and 64 are shown for the multi-threaded tools (KMC, SBWT default, and SBWT low-memory mode). The sorted-lists baseline is single-threaded. The $t64$ values correspond to the set-operation columns of Table A4. The baseline tools (KMC and sorted lists) perform ordered $k$-mer set manipulation, whereas SBWT set operations additionally construct a richer indexed representation supporting $k$-mer lookup and SBWT graph navigation, which explains the higher runtime and memory usage, and yields an order-of-magnitude smaller output index.

# On the Complexity of the $(\ell, k)$-Median Problems

**Luís Cunha**
Instituto de Computação, Universidade Federal Fluminense, Niterói, Brazil

**Thiago Nascimento**
Instituto de Computação, Universidade Federal Fluminense, Niterói, Brazil

**Marilia D. V. Braga**
Faculty of Technology and Center for Biotechnology (CeBiTec), Bielefeld University, Germany

**Jens Stoye**
Faculty of Technology and Center for Biotechnology (CeBiTec), Bielefeld University, Germany

## Abstract

The genome median problem is a central computational problem in comparative genomics, as it models the reconstruction of an ancestral genome from a set of related genomes. Given $\ell$ genomes and a distance measure, the problem asks for a genome that minimizes the sum of the distances to the input genomes. Two classical distances are the breakpoint distance and the double-cut-and-join (DCJ) distance. For multichromosomal circular genomes, the median problem is polynomial-time solvable under the breakpoint distance, whereas it is NP-hard under the DCJ distance.

For even integer $k \geq 2$, the $\sigma_k$ distance interpolates between these two extremes: $\sigma_2$ corresponds to the breakpoint distance, while $\sigma_\infty$ corresponds to the DCJ distance. A central open problem in this setting is the $(3, 4)$-Median problem, which asks for a median of three genomes under the $\sigma_4$ distance, the first intermediate distance after the breakpoint distance.

Motivated by this question, we study the more general $(\ell, k)$-Median problem, in which $\ell$ is the number of input genomes and $k$ determines the $\sigma_k$ distance. We prove that $(\ell, 6)$-Median for every $\ell \geq 4$ and $(3, 12)$-Median are NP-complete. We then extend the hardness of $(\ell, 6)$-Median to $(\ell, k)$-Median for all even $k \geq 6$, and the hardness of $(3, 12)$-Median to $(3, k)$-Median for all even $k \geq 12$. These results identify broad hardness regions in the $(\ell, k)$ parameter space and delimit the remaining open cases around the fundamental $(3, 4)$-Median problem.

**2012 ACM Subject Classification** Theory of computation → Proof complexity

**Keywords and phrases** Genome rearrangement, median problem, sigma-$k$ distance

**Digital Object Identifier** 10.4230/LIPIcs.WABI.2026.19

**Funding** *Luís Cunha*: FAPERJ: JCNE - E-26/201.372/2022 and E-26/204.301/2025; CNPq: PQ - 307666/2025-5, Universal - 406173/2021-4, National Institute of Science and Technology for Applied Quantum Computing - 408884/2024-0.
*Marilia D. V. Braga*: CAPES PRINT: 88887.310247/2018-00.
*Jens Stoye*: CAPES PRINT: 88887.310247/2018-00.

## 1 Introduction

Phylogenetics studies evolutionary relationships among species using genetic information obtained, for example, through DNA sequencing. A central task in this context is to infer ancestral genomes from a set of related extant genomes. Comparative genomics, in which genomes of different species are compared, helps address fundamental questions at the intersection of genetics and evolution [9].

From a computational perspective, ancestral reconstruction is naturally modeled by the *median problem* [13]. Given a set of $\ell \geq 3$ genomes and a distance measure, the median problem asks for a genome that minimizes the sum of its distances to the input genomes.


© Luís Cunha, Thiago Nascimento, Marilia D. V. Braga, and Jens Stoye;
licensed under Creative Commons License CC-BY 4.0

Although the case $\ell = 3$ is the classical formulation of the problem, the problem is naturally defined for any number of input genomes and has also been studied in this more general setting [8]. A median genome can be interpreted as a candidate ancestral genome for the input genomes.

The choice of the distance measure depends on the genome model. In this paper, a genome is represented as a set of *chromosomes*, and each chromosome is a circular sequence of oriented *genes* classified into gene *families*. Two genomes over the same set of genes form a *canonical pair* if they are *singular*, i.e., each genome contains exactly one gene from each family. One classical distance in this setting is the *breakpoint distance* [14], which counts the non-common adjacencies of two genomes, where an *adjacency* is the oriented neighborhood between two consecutive genes in a chromosome.

Genomes can also be restricted according to the structure of their chromosomes: they may be monochromosomal or multichromosomal, and their chromosomes may be linear, circular, or mixed. Different genomic distances have been studied for these scenarios, ranging from the breakpoint distance to the more general *double-cut-and-join* (DCJ) distance [16, 4].

The relationship between two genomes can be represented by the *breakpoint graph*. For canonical circular genomes, this graph is a collection of even cycles. We call a cycle with $i$ edges an *$i$-cycle*, and an *even cycle* if $i$ is even. Let $c_i$ be the number of $i$-cycles in the breakpoint graph. If $n$ is the number of genes in each genome, the breakpoint distance is $n - c_2$. Similarly, for rearrangements modeled by the DCJ operation, the DCJ distance is $n - c$, where $c$ is the total number of cycles.

The computational complexity of the median problem depends on both the genome model and the chosen distance. For circular genomes, the median problem is polynomial-time solvable under the breakpoint distance, but NP-hard under the DCJ distance [13].

A natural way to study the gap between these two extremes is through the family of $\sigma_k$ distances. For an even integer $k \geq 2$, let $\sigma_k = c_2 + c_4 + \cdots + c_k$ and define $\mathrm{d}_k = n - \sigma_k$ [6]. Thus, the distance defined by $\sigma_2$ corresponds to the breakpoint distance, while $\sigma_\infty$ corresponds to the DCJ distance.

The $\sigma_k$ family has also been studied in the context of the *double distance* problem, whose goal is to compute the minimum number of genomic rearrangements between a singular and a duplicated genome that – in addition to rearrangements – are separated by a whole genome duplication. For that problem, the cases $k = 2, 4, 6$ can be solved in linear time [5], whereas the problem is NP-complete for every $k \geq 8$ [7]. Together with the classical results for the median problem under the breakpoint and DCJ distances [13], these works motivate a finer investigation of median problems under intermediate $\sigma_k$ distances.

The main motivation for this work is the complexity of the $(3, 4)$-Median problem. This problem asks for a median of three genomes under the $\sigma_4$ distance and represents the first nontrivial intermediate case between the polynomial-time breakpoint median problem and the NP-hard DCJ median problem. The relevance of this question has been highlighted in previous work on genome rearrangement distances and median problems [13, 12].

Motivated by this question, we study the more general $(\ell, k)$-Median problem. This formulation allows us to investigate how the complexity depends on the number $\ell$ of input genomes and on the distance parameter $k$. Eventually, it may lead to better approximations to the (more relevant) DCJ median, while still staying at or close to the polynomial time complexity of the breakpoint median.

This paper is organized as follows. In Section 2, we introduce the necessary background on genomes, breakpoint graphs, and the $\sigma_k$ distance, and we formally define the $(\ell, k)$-Median problem. In Section 3 we introduce concepts and properties that are common to our

reductions, including the construction of multiple breakpoint graphs from regular digraphs. In Section 4, starting from the NP-complete problem *Maximum-cardinality edge-disjoint directed triangle decomposition*, we present our results for the smallest values of $k$: We first prove the NP-completeness of the median problem for the cases $\ell \geq 4$ and $k = 6$ and for the case $\ell = 3$ and $k = 12$. In Section 5, we extend these results to larger values of $k \geq 8$ for $\ell \geq 4$ and of $k \geq 14$ for $\ell = 3$. Table 1 summarizes the complexity landscape for $(\ell, k)$-Median and highlights the results proved in this paper.

**Table 1** Complexity of the multichromosomal circular $(\ell, k)$-Median problem. Each row represents the number of input genomes, and each column represents the parameter $k$ used in the $\sigma_k$ distance. Results proved in this paper are shown in blue.

| # of genomes \ $\sigma_k$ distance | 2 | 4 | 6 | 8 | 10 | 12 | $k \geq 14$ | $\infty$ |
|---|---|---|---|---|---|---|---|---|
| 3 | P [13] | Open | Open | Open | Open | NP-hard | NP-hard | NP-hard (DCJ) [13] |
| 4 | P [13] | Open | NP-hard | NP-hard | NP-hard | NP-hard | NP-hard | NP-hard (DCJ) [13] |
| $\ell \geq 5$ | P [13] | Open | NP-hard | NP-hard | NP-hard | NP-hard | NP-hard | NP-hard (DCJ) [13] |

## 2 Preliminaries

A genome is a set of *chromosomes*. In this work, all chromosomes are circular, representing circular DNA molecules, and are modeled as oriented circular sequences of *genes*. Each gene belongs to a *gene family*. If all genes in a genome are from different gene families, the genome is called *singular*. A gene $\mathtt{X}$ has two extremities, denoted by $\mathtt{X}^t$ and $\mathtt{X}^h$, which correspond to its tail and its head, respectively. The gene is written as $\mathtt{X}$ when read in its forward orientation, from tail to head, and as $\bar{\mathtt{X}}$ when read in its reverse orientation, from head to tail. A chromosome is represented by a sequence of genes enclosed in parentheses, for example $(\bar{3}\ 2\ \bar{1})$.

An *adjacency* in a chromosome is an unordered pair of neighboring gene extremities. For instance, the chromosome $(\bar{3}\ 2\ \bar{1})$ has $\{3^t2^t, 2^h1^h, 1^t3^h\}$ as its adjacencies. Since an adjacency is an unordered pair, $3^t2^t$ and $2^t3^t$ represent the same adjacency.

A graphical representation of a singular genome $\mathbb{G}$ is the *genome graph* whose vertex set contains two vertices for each gene $\mathtt{X}$ of $\mathbb{G}$, one representing its tail $\mathtt{X}^t$ and one representing its head $\mathtt{X}^h$, and whose edge set contains the adjacencies of $\mathbb{G}$: for each adjacency $AB$, there is an edge (called $\mathbb{G}$-*edge*) connecting $A$ and $B$. Every vertex is therefore incident to one $\mathbb{G}$-edge and the circular genome $\mathbb{G}$ represents a perfect matching.

While the genome graph is just an alternative representation of a genome, the relationships between two or more *canonical genomes*, i.e., singular genomes over the same set of genes, can be represented by a structure obtained by merging their genome graphs:

▶ **Definition 1** (Multiple breakpoint graph). *Given $\ell$ canonical genomes $\mathbb{G}_1, \mathbb{G}_2, \ldots, \mathbb{G}_\ell$, the* multiple breakpoint graph *$MBG(\mathbb{G}_1, \mathbb{G}_2, \ldots, \mathbb{G}_\ell) = (V, E)$ is constructed as follows: The vertex set $V$ contains two vertices for each gene $\mathtt{X}$, one representing its tail $\mathtt{X}^t$ and one representing its head $\mathtt{X}^h$. The edge multiset $E$ contains the adjacencies of $\mathbb{G}_1, \mathbb{G}_2, \ldots, \mathbb{G}_\ell$: For each adjacency $AB$ in genome $\mathbb{G}_i$, there is a $\mathbb{G}_i$-edge connecting $A$ and $B$.*

The multiple breakpoint graph of $\ell = 2$ genomes is also called just *breakpoint graph*. Since each circular genome corresponds to a perfect matching, the breakpoint graph decomposes into a set of cycles alternating between edges of the two genomes. We denote by $c_i$ the number of such cycles of length $i$, where integer $i$ is obviously even.

The breakpoint distance between two canonical genomes $\mathbb{G}_1$ and $\mathbb{G}_2$ can be derived from their breakpoint graph. Assuming that the genomes have $n$ genes each, the corresponding breakpoint distance is $d_{BP}(\mathbb{G}_1, \mathbb{G}_2) = n - c_2$. Similarly, the DCJ distance is $d_{DCJ}(\mathbb{G}_1, \mathbb{G}_2) = n - c$, where $c$ is the total number of cycles in $MBG(\mathbb{G}_1, \mathbb{G}_2)$. Equivalently, it can be written as $d_{DCJ}(\mathbb{G}_1, \mathbb{G}_2) = n - (c_2 + c_4 + \cdots + c_\infty)$.

The intermediate region between these two extremes can be studied through the family of $\sigma_k$ *distances*. For any even integer $k \geq 2$, let $\sigma_k = c_2 + c_4 + \cdots + c_k$. Then, given two canonical genomes $\mathbb{G}_1$ and $\mathbb{G}_2$ over the same set of $n$ genes $\mathcal{G}$ and an even integer $k \geq 2$, their $\sigma_k$ distance, denoted by $\mathrm{d}_k(\mathbb{G}_1, \mathbb{G}_2)$, is defined as $\mathrm{d}_k(\mathbb{G}_1, \mathbb{G}_2) = n - \sigma_k$ [6]. Thus, the breakpoint distance corresponds to $d_2$, while the DCJ distance corresponds to $d_\infty$.

In this work, we study the following problem:

▶ **Problem 2** ($(\ell, k)$-Median). *Given a set of $\ell \geq 3$ canonical circular genomes $\mathbb{G}_1, \mathbb{G}_2, \ldots, \mathbb{G}_\ell$ over the same set of $n$ genes $\mathcal{G}$ and an even integer $k \geq 2$, find a circular genome $\mathbb{M}$ over $\mathcal{G}$ that is canonical with the input genomes and minimizes the sum $\sum_{i=1}^{\ell} \mathrm{d}_k(\mathbb{G}_i, \mathbb{M})$.*

Observe that, like the input circular genomes, the circular median genome $\mathbb{M}$ also corresponds to a perfect matching in the multiple breakpoint graph. Since the number of genes is fixed at $n$ for all canonical genomes in the instance, minimizing the sum of distances is equivalent to maximizing the *$k$-score* $\sum_{i=1}^{\ell} \sigma_k(\mathbb{G}_i, \mathbb{M})$. In other words, the goal is to maximize the total number of alternating cycles in the multiple breakpoint graph, counted between the median genome $\mathbb{M}$ and the input genomes $\mathbb{G}_i$, for $i \in \{1, \ldots, \ell\}$. A cycle between the median and genome $\mathbb{G}_i$ is called *$\mathbb{G}_i$-cycle*. In our illustrations of multiple breakpoint graphs, each genome is represented by a distinct color. Therefore, we may refer to the genome $\mathbb{G}_i$ or to the color $\mathbb{G}_i$ interchangeably.

We focus on the *decision version* of the problem, which is the one used for complexity analysis: Given an instance as above and an integer threshold $T$, the question is whether there exists a genome $\mathbb{M}$ such that $\sum_{i=1}^{\ell} \mathrm{d}_k(\mathbb{G}_i, \mathbb{M}) \leq T$, or equivalently, $\sum_{i=1}^{\ell} \sigma_k(\mathbb{G}_i, \mathbb{M}) \geq \ell n - T$.

## 3 Basic tools

In this section we derive basic properties of multiple breakpoint graphs and median genomes, and introduce general gadgets that we will use in our reductions.

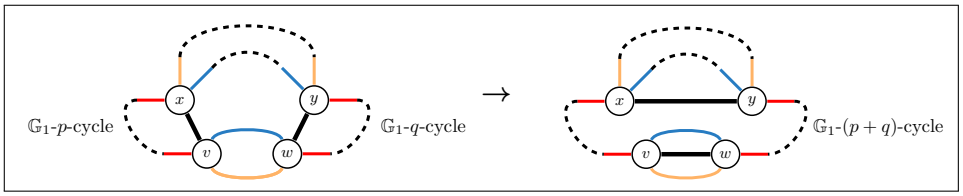
### 3.1 Subdivision gadget and its gates

We first demonstrate an extension of a property that has been used in other works concerning the median problem [15, 10, 12]. Let a $\widetilde{\mathbb{G}}_i$-adjacency $vw$ be an adjacency occurring in all genomes but $\mathbb{G}_i$.

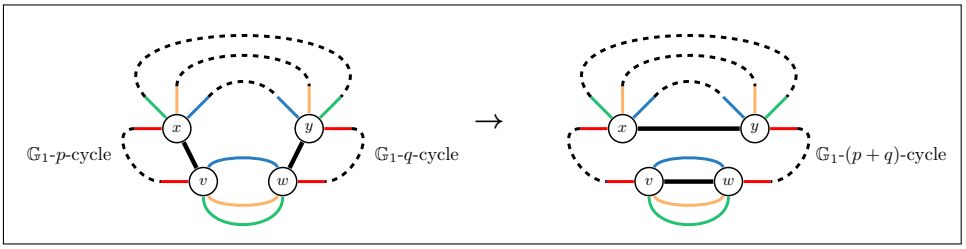
▶ **Lemma 3.** *Let $\mathbb{G}_i$ be one genome of an instance of $(\ell, k)$-median, for some $\ell \geq 3$ and some $k \geq 2$, and let $vw$ be a $\widetilde{\mathbb{G}}_i$-adjacency. It holds that $vw$ occurs in at least one median if $\ell = 3$ and in all medians if $\ell \geq 4$.*

**Proof.** Let the $\ell$ input genomes be $\mathbb{G}_1$, $\mathbb{G}_2$, …, $\mathbb{G}_\ell$. We assume that $vw$ occurs in genomes $\mathbb{G}_2, \ldots, \mathbb{G}_\ell$ and not in $\mathbb{G}_1$. Clearly, for the vertices $v$ and $w$, either $vw \in \mathbb{M}$ or $vw \notin \mathbb{M}$. Assume that $vw \notin \mathbb{M}$ and, since $\mathbb{M}$ is a perfect matching, let $x$ and $y$ be vertices such that $vx, wy \in \mathbb{M}$; see Figure 1 (a) for $\ell = 3$ and Figure 1 (b) for $\ell = 4$. In this case $\mathbb{M}$ forms paths with genomes $\mathbb{G}_2$, …, $\mathbb{G}_\ell$, and each of them is part of a ($\mathbb{G}_2$- or … or $\mathbb{G}_\ell$-) cycle of length at least 4. Furthermore, $\mathbb{M}$ forms two paths with genome $\mathbb{G}_1$, that could in the best case

be part of two $\mathbb{G}_1$-cycles, each one of length at least 2. By replacing the edges $vx, wy$ with $vw, xy$ in $\mathbb{M}$, we split each of the $\mathbb{G}_2$-, ..., $\mathbb{G}_\ell$-cycles into two smaller cycles, one being a 2-cycle, which would increase the $k$-score by $\ell - 1$. This move would merge the two possible $\mathbb{G}_1$-cycles, possibly decreasing the score by 2. Therefore, if $\ell = 3$, the $k$-score would certainly not decrease, and if $\ell \geq 4$, the $k$-score would even certainly increase. ◀




**(a)** $\ell = 3$.

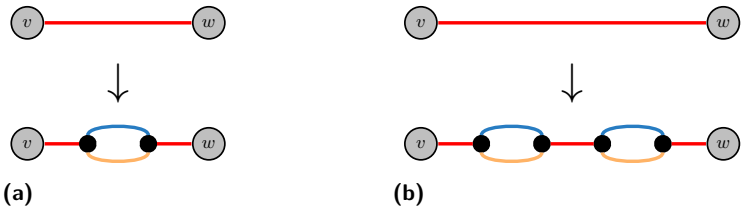



**(b)** $\ell = 4$.

**Figure 1** Illustration of the proof of Lemma 3, with median edges drawn in black. It shows a local part of the multiple breakpoint graph assuming $vw \notin \mathbb{M}$, followed by the same local part after replacing $vx, wy$ with $vw, xy$ in $\mathbb{M}$ for **(a)** $\ell = 3$ and **(b)** $\ell = 4$.

Let $v$ and $w$ be vertices of graph $H = MBG(\mathbb{G}_1, \mathbb{G}_2, \ldots, \mathbb{G}_\ell)$ and let $i$ be an integer with $1 \leq i \leq \ell$. If $vw$ is a $\widetilde{\mathbb{G}}_i$-adjacency in $H$, then we say that vertices $v$ and $w$ form a *Z-pair*.

We build a general gadget that can be defined for a set of $\ell$ colors as follows. The *$\ell$-subdivision* gadget, denoted by $Z_\ell$, splits a $\mathbb{G}_i$-edge $vw$ and inserts two new vertices $v'$ and $w'$, such that $vv'$ and $ww'$ are $\mathbb{G}_i$-edges and $v'w'$ is a $\widetilde{\mathbb{G}}_i$-adjacency. We then extend it to $Z_\ell^q$, the *$\ell^q$-subdivision* of a $\mathbb{G}_i$-edge $uv$ for an integer $q \geq 1$: This operation inserts $q$ $\widetilde{\mathbb{G}}_i$-adjacencies in $vw$. Since $vw$ is a $\mathbb{G}_i$-edge, we can say that its corresponding $Z_\ell^q$-gadget is of color $\mathbb{G}_i$. The outermost vertices of $Z_\ell^q$ are still vertices $v$ and $w$, and are now called *gates*, while the (new) inner vertices of $Z_\ell^q$ are distributed into $Z$-pairs. Note that $Z_\ell^1 = Z_\ell$. Examples of $Z_3$ and $Z_3^2$ are shown in Figure 2.




**(a)** **(b)**

**Figure 2** Let $i$ be an integer between 1 and 3. **(a)** A $Z_3^1$- or simply $Z_3$-gadget splits a $\mathbb{G}_i$-edge $vw$ and inserts a $\widetilde{\mathbb{G}}_i$-adjacency (a pair of vertices connected by two parallel edges in the two other colors). **(b)** A $Z_3^2$-gadget inserts two $\widetilde{\mathbb{G}}_i$-adjacencies into a $\mathbb{G}_i$-edge $vw$. In both (a) and (b), the two endpoints of each $\widetilde{\mathbb{G}}_i$-adjacency are $Z$-pairs, and vertices $v$ and $w$ are the gates of the gadget.

## 3.2 The $\alpha$-color and $\Theta$-transformation of $p$-regular digraphs

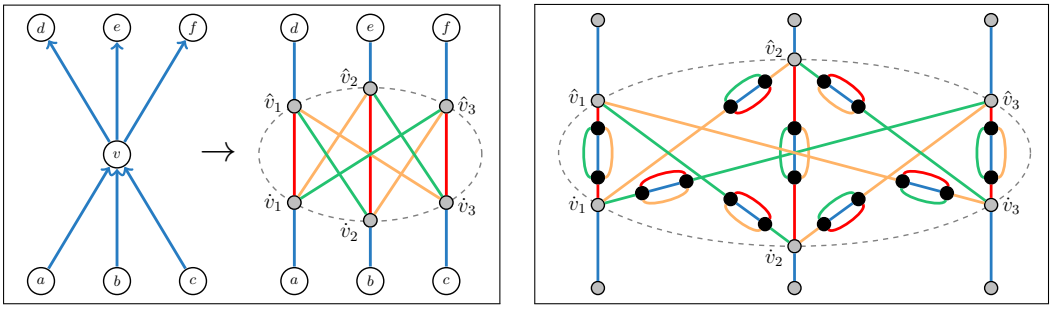
In our reductions, we construct edge $\ell$-colored graphs in which $\ell-1$ colors, named $\mathbb{G}_1, \mathbb{G}_2, \ldots, \mathbb{G}_{\ell-1}$, are symmetrically distributed, while the $\ell$th color has a special role. Therefore, instead of calling it $\mathbb{G}_\ell$ we call the color $\mathbb{G}_\alpha$ or simply $\alpha$. We denote the edges of this color $\mathbb{G}_\alpha$-edges or simply $\alpha$-edges, and its median cycles $\mathbb{G}_\alpha$-cycles or simply $\alpha$-cycles.

In the following parts of this manuscript, a color $\mathbb{G}_i$ refers to some color that is not $\alpha$, that is, $\mathbb{G}_i \neq \alpha$, therefore any set $\widetilde{\mathbb{G}}_i$ (of all colors but $\mathbb{G}_i$) always includes $\alpha$. We denote by $\mathbb{G}_{\hat{\alpha}}$ any unspecified color that is not $\alpha$.

We start from directed graphs (*digraphs*). A digraph is *p-regular* if every vertex has both in-degree and out-degree equal to $p$ [3]. We will now explain how to construct, from a $p$-regular digraph $D$ and an integer $q \geq 1$, an undirected $(p+1)$-colored graph $\Theta(D, q)$ that includes $(p+1)^q$-subdivision gadgets.

▶ **Construction 4** ($\Theta$-transformation). *Given a $p$-regular digraph $D = (V, E)$ (for $p \geq 2$) and an integer $q \geq 1$, we construct the undirected edge $(p+1)$-colored graph $\Theta(D, q)$ as follows:*

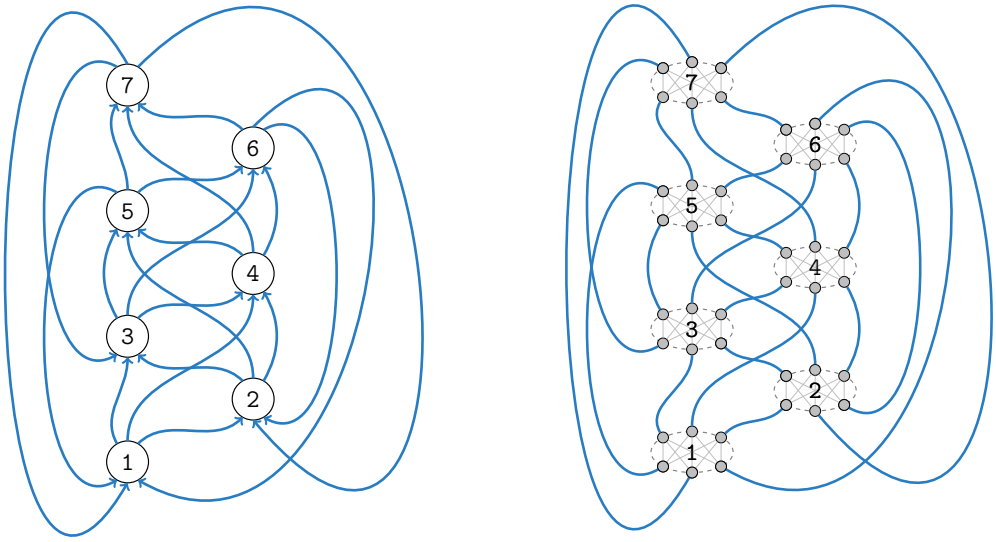
1. *(Vertex $2p$-expansions with $\mathbb{G}_{\hat{\alpha}}$-edges) For each vertex $v \in V$, create a $p$-colored undirected complete bipartite graph $B^p(u)$ whose partitions are $\{\dot{v}_1, \dot{v}_2, \ldots, \dot{v}_p\}$ (*in-plugs*) and $\{\hat{v}_1, \hat{v}_2, \ldots, \hat{v}_p\}$ (*out-plugs*). The edges of $B^p(v)$ are distributed into $p$ disjoint perfect matchings, and each matching has one of the $p$ $\mathbb{G}_{\hat{\alpha}}$-colors $\mathbb{G}_1, \mathbb{G}_2, \ldots, \mathbb{G}_p$ as shown in Figure 3(a) for $p = 3$.*
2. *($q$-Subdivisions of $\mathbb{G}_{\hat{\alpha}}$-edges) For each vertex $v \in V$, by a transformation of $B^p(v)$, where each of its $p^2$ $\mathbb{G}_{\hat{\alpha}}$-edges is subdivided by the insertion of a $Z^q_{p+1}$-gadget whose additional color is $\alpha$, we obtain the gadget $X^{p,q}(v)$ (see Figure 3(b) for $p = 3$ and $q = 1$). Note that each in-plug $\dot{v}$ and each out-plug $\hat{v}$ of $B^p(v)$ becomes a gate of $p$ $Z^q_{p+1}$-gadgets in $X^{p,q}(v)$ (of all $p$ distinct $\mathbb{G}_{\hat{\alpha}}$-colors).*




**(a)** From a vertex $v$ of $D$ to $B^3(v)$. **(b)** Gadget $X^{3,1}(v)$.

**Figure 3** Construction of gadget $X^{3,1}(v)$. **(a)** Vertex expansion: Each vertex $v$ of 3-regular digraph $D$ is replaced by a complete undirected bipartite graph $B^3(v)$. The edges are split into 3 perfect matchings, each one receives one color $\mathbb{G}_1$, $\mathbb{G}_2$ or $\mathbb{G}_3$ (here red, green and orange). The original (directed) edges become undirected $\alpha$-edges (here blue). **(b)** Subdivision: The gadget $X^{3,1}(v)$ is obtained from $B^3(v)$ by replacing each of its internal edges by the corresponding $Z_4$-gadget. From the six vertices $a, b, c, d, e, f$ adjacent to $v$ we only show the respective plugs connected by $\alpha$-edges to the plugs of $X^{3,1}(v)$.

3. *(Connection between gadgets with $\alpha$-edges) For each vertex $v \in V$, for each out-edge $(v \to w)$, create in $\Theta(D, q)$ an undirected $\alpha$-edge connecting one of the out-plugs of $X^{p,q}(v)$ to one of the in-plugs of $X^{p,q}(w)$. Similarly, for each in-edge $(u \to v)$, create an undirected $\alpha$-edge connecting one of the out-plugs of $X^{p,q}(u)$ to one of the in-plugs of $X^{p,q}(v)$ in $\Theta(D, q)$. These connections should be done such that exactly one $\alpha$-edge is incident to each plug (Figure 4).*




**(a)** **(b)**

**Figure 4** Creation of the undirected, 4-colored graph $\Theta(D, q)$ from a 3-regular digraph $D$ and $q \geq 1$. **(a)** A 3-regular digraph $D$. **(b)** The graph $\Theta(D, q)$ obtained from $D$ via Construction 4, where each $X^{3,q}$-gadget is represented by a dashed ellipse. Colors $\mathbb{G}_1, \mathbb{G}_2$, and $\mathbb{G}_3$ are present only in edges inside the $X^{3,q}$-gadgets, masked by the ellipses. Color $\alpha$ (here blue) is a union of the $\alpha$-edges between ellipses and the $\alpha$-edges present in all $Z_4^q$-gadgets, the latter being also masked by the ellipses.

▶ **Proposition 5.** *If graph $\Theta(D, q)$ is obtained via Construction 4 on $p$-regular digraph $D$ and an integer $q \geq 1$, then its $p+1$ colors define $p+1$ circular genomes $\mathbb{G}_1$, $\mathbb{G}_2$, ..., $\mathbb{G}_p$ and $\mathbb{G}_\alpha$ such that $\Theta(D, q) = MBG(\mathbb{G}_1, \mathbb{G}_2, \ldots, \mathbb{G}_p, \mathbb{G}_\alpha)$.*

**Proof.** Note that $\Theta(D, q)$ has an even number of vertices: Each $X^{p,q}$-gadget has an even number of plugs and each $Z_{p+1}^q$ gadget adds $q$ $Z$-pairs. Furthermore, it is easy to see that each of the vertices of $\Theta(D, q)$ is incident to exactly $p+1$ edges, each one of a distinct color $\mathbb{G}_1$, $\mathbb{G}_2$, ..., $\mathbb{G}_p$ or $\mathbb{G}_\alpha$. This guarantees that each of the $p+1$ colors defines a perfect matching of the vertices of $\Theta(D, q)$, therefore we can say that each color defines the adjacencies of one circular genome. The $p+1$ genomes are then named $\mathbb{G}_1$, $\mathbb{G}_2$, ..., $\mathbb{G}_p$ and $\mathbb{G}_\alpha$. ◀

## 3.3 Median partitions $\mathbb{M}_Z$ and $\mathbb{M}_*$, $\mathbb{G}_{\hat{\alpha}}$-bridges and $\alpha$-shortcuts

Due to Lemma 3, we consider only medians for $\Theta(D, q)$ that include an edge connecting the two vertices of each of its $Z$-pairs.

▶ **Proposition 6.** *Let $k \geq 2$ and $\Theta(D, q)$ be a graph obtained via Construction 4 on a $p$-regular digraph $D = (V, E)$ and an integer $q \geq 1$. There is a median $\mathbb{M}$ for $\Theta(D, q)$ that can be partitioned into sets $\mathbb{M}_Z$ and $\mathbb{M}_*$, where $\mathbb{M}_Z$ has one edge connecting each $Z$-pair and $\mathbb{M}_*$ is a perfect matching of the plugs of $\Theta(D, q)$. Furthermore, the $k$-score of $\mathbb{M}$ is $s_Z + s_*$, where the first term, also denoted $Z$-score, is a constant $s_Z = qp^3|V|$, while the second term, $s_*$, is the maximum $k$-score attainable by connecting the plugs of $\Theta(D, q)$.*

**Proof.** The fact that there exists a median $\mathbb{M}$ for $\Theta(D, q)$ that includes an edge for each $Z$-pair in $\Theta(D, q)$ is a direct consequence of Lemma 3. Each of the $Z$-pairs induces $p$ 2-cycles, that contribute to any $k$-score for $k \geq 2$. After fixing $\mathbb{M}_Z$, there is no "free" vertex in $\Theta(D, q)$

that is not a plug, therefore $\mathbb{M}_*$ must be a perfect matching of the plugs. Since there are $p^2$ $Z^q_{p+1}$-gadgets for each vertex of $D$, and each one has $pq$ $Z$-pairs, they yield the overall score $s_Z = qp^3|V|$, which is part of the $k$-score of any optimal solution that includes set $\mathbb{M}_Z$. ◀

All edges of $\Theta(D, q)$ (from all defined $p+1$ colors) that form 2-cycles with $\mathbb{M}_Z$ cannot be part of any cycle induced by $\mathbb{M}_*$ and are therefore said to be *$Z$-saturated*.

Note that each $\mathbb{G}_i$-adjacency in $\Theta(D, q)$ is either $Z$-saturated or part of a $Z^q_\ell$-gadget $Y$ of color $\mathbb{G}_i$. Since $Y$ is a chain of $\mathbb{G}_i$-edges that alternate with $\widetilde{\mathbb{G}}_i$-adjacencies, together with edges from $\mathbb{M}_Z$, it induces a fixed $\mathbb{G}_i$-path of length $2q+1$. Such a path is called *$\mathbb{G}^q_i$-bridge*, whose endpoints (gates) are necessarily plugs of $\Theta(D, q)$. If $q = 1$ we can use the simpler notation $\mathbb{G}_i$-bridge. An example is given in Figure 5 (a). Since any $\mathbb{G}_{\hat{\alpha}}$-edge that is not $Z$-saturated is part of a bridge, we have:

▶ **Observation 7.** *Let $\mathbb{G}_i$ be a $\mathbb{G}_{\hat{\alpha}}$-color and recall that each $\mathbb{G}^q_i$-bridge is a $(2q+1)$-path. Any $\mathbb{G}_i$-cycle induced by $\mathbb{M}_*$ alternates median edges (from $\mathbb{M}_*$) with $\mathbb{G}^q_i$-bridges and has therefore length $b(2q+1+1) = 2b(q+1)$, where $b \geq 1$ is the number of involved $\mathbb{G}^q_i$-bridges.*

The idea is then to set the value of $q$ such that a median cycle involving a single bridge has length at most $k$, while a cycle involving two or more bridges has length greater than $k$. For any $k \geq 4$, the smallest $q$ that fulfills this requirement is $q = \lfloor \frac{k}{4} \rfloor$: if $b = 1$, then $2b\left(\lfloor \frac{k}{4} \rfloor + 1\right) = 2\left(\lfloor \frac{k}{4} \rfloor + 1\right) \leq k$. If $b \geq 2$, then $2b\left(\lfloor \frac{k}{4} \rfloor + 1\right) \geq 4\left(\lfloor \frac{k}{4} \rfloor + 1\right) > k$. Table 2 displays the behavior of $q$ as $k$ grows. In this setting, only $\mathbb{G}_{\hat{\alpha}}$-cycles involving a single bridge contribute to the $k$-score.

**Table 2** For each possible $k \geq 4$ we show: the value of $q$, the length of $\mathbb{G}^q_{\hat{\alpha}}$-bridges induced by $\mathbb{M}_Z$ and the lengths of $\mathbb{G}_{\hat{\alpha}}$-cycles induced by $\mathbb{M}_*$.

| $k$ | 4 or 6 | 8 or 10 | 12 or 14 | 16 or 18 | 20 or 22 | $k \geq 24$ |
|---|---|---|---|---|---|---|
| $q$ | 1 | 2 | 3 | 4 | 5 | $q = \lfloor \frac{k}{4} \rfloor$ |
| length of all $\mathbb{G}^q_{\hat{\alpha}}$-bridges | 3 | 5 | 7 | 9 | 11 | $2q+1$ |
| length of a $\mathbb{G}_i$-cycle involving $b$ $\mathbb{G}^q_i$-bridges | $4b$ | $6b$ | $8b$ | $10b$ | $12b$ | $2b(q+1)$ |

Such a cycle is obtained when an edge in $\mathbb{M}_*$, which is then called *cross edge*, connects plugs that are the two endpoints (gates) of the same $\mathbb{G}^q_i$-bridge. In each $X^{p,q}$-gadget at most $p$ $\mathbb{G}_{\hat{\alpha}}$-cycles of length $2(q+1) \leq k$ can be built with cross edges. (Figure 5 (b) illustrates this scenario for a $X^{3,1}$-gadget.)

If an edge of $\Theta(D, q)$ is not $Z$-saturated nor part of a bridge, it must be an $\alpha$-edge connecting plugs and is then called *$\alpha$-shortcut*. Observe that $\Theta(D, q)$ has $2p|V|$ plugs, thus any perfect matching of the plugs has $p|V|$ edges. The $\alpha$-shortcuts form a perfect matching of the plugs such that each $\alpha$-shortcut connects an in- and an out-plug of distinct $X^{p,q}$-gadgets.

An edge in $\mathbb{M}_*$ is called *loop edge* if it connects plugs that are the two endpoints of the same $\alpha$-shortcut. A median edge that is not a loop edge nor a cross edge is called *jump edge*. Examples are given in Figure 6 (a).

▶ **Observation 8.** *An oriented $t$-cycle $C$ in $p$-regular digraph $D$ implies that, for each edge $(v \to w)$ in $C$, the graph $\Theta(D, q)$ has an $\alpha$-shortcut connecting an out-plug $\hat{v}$ of $X^{p,q}(v)$ to an in-plug $\hat{w}$ of $X^{p,q}(w)$. When $\mathbb{M}_*$ includes the cross edges chaining the endpoints of all $\alpha$-shortcuts derived from $t$-cycle $C$, then $\mathbb{M}_*$ induces an $\alpha$-$2t$-cycle in $\Theta(D, q)$.*

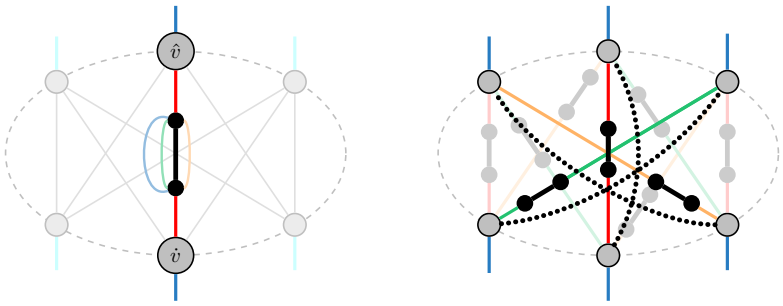



**Figure 5** The (fixed) median edges from partition $\mathbb{M}_Z$ are represented as solid black, while (candidate) median edges from partition $\mathbb{M}_*$ are represented as dotted black. **(a)** Plugs $\dot{v}$ and $\hat{v}$ are the gates of a $Z_4^1$ gadget of color $\mathbb{G}_1$ (red). Median partition $\mathbb{M}_Z$ includes the (black) median edge connecting the pair of black vertices (called $Z$-pair). The shaded edges of colors $\mathbb{G}_2$, $\mathbb{G}_3$ and $\alpha$ (here green, orange and blue) connecting the $Z$-pair of black vertices are $Z$-saturated: They form 2-cycles with the median edge. Together with the (red) $\mathbb{G}_1$-edges, the median edge also forms a $\mathbb{G}_1$-bridge, that is the alternating path of length 3 connecting $\dot{v}$ to $\hat{v}$. **(b)** Here we omit $Z$-saturated edges, but all inner connections are displayed: Between each in-plug and each out-plug there is a $Z_4^1$-gadget of some color $\mathbb{G}_i$ that gives a $\mathbb{G}_i$-brigde of length 3. We can then have up to $p = 3$ $\mathbb{G}_{\hat{\alpha}}$-cycles, each of length 4, with (dotted black) median cross edges from partition $\mathbb{M}_*$. Here each of these cycles is with a distinct $\mathbb{G}_i$, but in general they can have other distributions among the $p$ $\mathbb{G}_i$s.

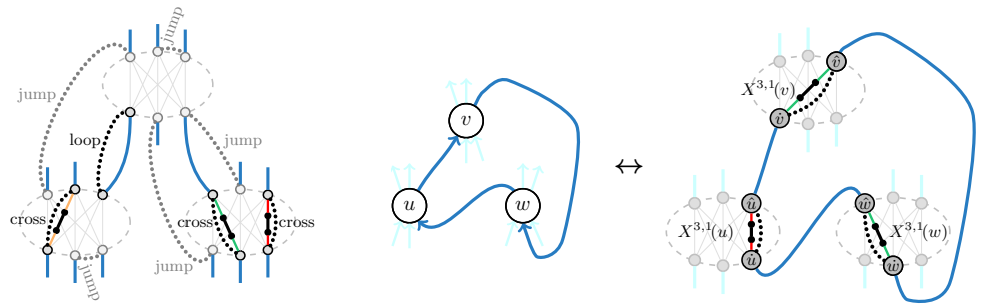



**Figure 6** In these pictures, $Z$-saturated edges are omitted. The (fixed) median edges from partition $\mathbb{M}_Z$ are represented as solid black, while (candidate) median edges from partition $\mathbb{M}_*$ are represented as dotted black or gray. **(a)** Section of a graph showing three $X^{3,1}$-gadgets. One of the black dotted edges is a median loop edge, parallel to a (blue) $\alpha$-shortcut. The other three black dotted edges are median cross edges, that are parallel to $\mathbb{G}_{\hat{\alpha}}$-bridges. All gray dotted edges are median jump edges: either they connect two (bottom) in- or two (top) out-plugs of the same or of distinct $X^{3,1}$-gadgets. Another type of jump edge connects a (bottom) in-plug and a (top) out-plug of distinct $X^{3,1}$-gadgets but is not parallel to an $\alpha$-shortcut. **(b)** A directed 3-cycle $(uvw)$ in a 3-regular digraph $D$ and the potential corresponding 6-cycle in $\Theta(D, 1)$: The directed edge $(u \to v)$ in $D$ implies that there is an $\alpha$-shortcut $\hat{u}\dot{v} \in \Theta(D, 1)$ for some out-plug $\hat{u}$ of $X^{3,1}(u)$ and some in-plug $\dot{v}$ of $X^{3,1}(v)$. Similarly, the directed edges $(v \to w)$ and $(w \to u)$ imply that there are $\alpha$-shortcuts $\hat{v}\dot{w}$ and $\hat{w}\dot{u} \in \Theta(D, 1)$. The $\alpha$-shortcuts are colored in blue. If (dotted black) cross edges $\dot{u}\hat{u}$, $\dot{v}\hat{v}$ and $\dot{w}\hat{w}$ are in $\mathbb{M}_*$, we have an $\alpha$-6-cycle $(\dot{u}\hat{u}\dot{v}\hat{v}\dot{w}\hat{w})$. Each of the three median cross edges shown here also induces a $\mathbb{G}_{\hat{\alpha}}$-cycle of length 4.

Therefore, only $\alpha$-cycles that exclusively alternate $\alpha$-shortcuts with median cross edges in $\Theta(D, q)$ can correspond to oriented cycles in $D$. An example of a 3-cycle in a 3-regular graph D corresponding to an $\alpha$-6-cycle in $\Theta(D, 1)$ is given in Figure 6 (b).

## 4 NP-completeness results for the minimum $k$

In this section we will show our results with smallest $k$, which are the NP-completeness of the $(\ell, 6)$-Median problem for any $\ell \geq 4$ and the NP-completeness of the $(3, 12)$-Median problem.

### 4.1 NP-completeness of the $(\ell, 6)$-Median problem for $\ell \geq 4$

We will consider a restricted version of the NP-complete problem *maximum-cardinality edge-disjoint directed triangle decomposition*, described as follows: For an integer $p \geq 3$, the input is a $p$-regular digraph $D$ with no 2-cycles, and the goal is to find a maximum-cardinality collection of edge-disjoint directed 3-cycles in $D$. This restricted version is denoted MAX3CDEC and remains NP-complete [1].

Here we have $k = 6$, therefore $q = \lfloor \frac{6}{4} \rfloor = 1$. Let $\ell \geq 4$ and $p = \ell - 1$, let $D = (V, E)$ be a $p$-regular digraph, and let graph $\Theta(D, 1)$ be obtained via Construction 4. Observation 8 guarantees that an oriented 3-cycle in graph $D$ corresponds to a potential $\alpha$-6-cycle induced by set $\mathbb{M}_*$ in $\Theta(D, 1)$. Recall that, since for $q = 1$ the length of any $\mathbb{G}_i$-cycle is a multiple of 4 (Table 2), a $\mathbb{G}_i$-cycle can only contribute to the 6-score if it involves a single $\mathbb{G}_i$-bridge.

▶ **Proposition 9.** *There is a median $\mathbb{M}$ of $\Theta(D, 1)$ that is of the form $\mathbb{M} = \mathbb{M}_Z \cup \mathbb{M}_*$ and is free of jump edges.*

**Proof.** The choice of including a jump edge $e$ in $\mathbb{M}_*$ could always be replaced by another one using the same plugs, but with at least the same 6-score: Let us suppose that $e$ is part of an $\alpha$-4-cycle $c_4^\alpha$, whose second median edge is $e'$. If $e'$ is a cross edge, the cycle $c_4^\alpha$ coexists with a $\mathbb{G}_{\hat{\alpha}}$-4-cycle. Otherwise $e'$ is also a jump edge and $c_4^\alpha$ cannot co-exist with any $\mathbb{G}_{\hat{\alpha}}$-4-cycle. Replacing $e$ and $e'$ by two loop edges that are parallel to the $\alpha$-shortcuts of $c_4^\alpha$ would produce two $\alpha$-2-cycles, occupying the same four plugs. Although this new arrangement certainly prevents the incident bridges to form $\mathbb{G}_{\hat{\alpha}}$-4-cycles, it still has at least the same 6-score. Now let $e$ be part of an $\alpha$-cycle $c_m^\alpha$ with length $m \geq 6$. Note that $c_m^\alpha$ co-exists with at most $\frac{m}{2} - 1$ $\mathbb{G}_{\hat{\alpha}}$-4-cycles. Then it is clear that $c_m^\alpha$ could be replaced by $\frac{m}{2}$ $\alpha$-2-cycles using the same plugs, achieving at least the same 6-score if $m = 6$ and a higher 6-score if $m \geq 8$. ◀

We will now relate solutions of MAX3CDEC to median solutions of the instance represented by $\Theta(D, 1) = MBG(\mathbb{G}_1, \mathbb{G}_2, ..., \mathbb{G}_p, \mathbb{G}_\alpha)$.

▶ **Theorem 10.** *The problem $(\ell, 6)$-Median is NP-complete for any $\ell \geq 4$.*

**Proof.** Take the subdivision factor $q = \lfloor \frac{6}{4} \rfloor = 1$. For $p = \ell - 1$, let $D$ be a $p$-regular digraph free of 2-cycles and let $\Theta(D, 1)$ be a graph obtained via Construction 4. Proposition 6 guarantees that there is a median $\mathbb{M}$ for $\Theta(D, 1)$ of the form $\mathbb{M} = \mathbb{M}_Z \cup \mathbb{M}_*$, where $\mathbb{M}_Z$ is fixed. Furthermore, the 6-score of $\mathbb{M}$ is $s_Z + s_*$, and $s_Z = qp^3|V|$, that gives $s_Z = p^3|V|$ for $q = 1$. We prove the following equivalence: An optimal solution of MAX3CDEC for a digraph $D = (V, E)$ contains $f$ directed 3-cycles if and only if $s_* = f + p|V|$.

First we show that an optimal solution $\mathcal{S}$ of MAX3CDEC in $D$ with $f$ directed 3-cycles yields a set $\mathbb{M}_*$ for which $s_* = f + p|V|$.

For each oriented 3-cycle $C = (uvw)$ in the solution $\mathcal{S}$, we can add the median cross edges to $\mathbb{M}_*$ so that we build an $\alpha$-6-cycle as described in Observation 8. After doing that for each of the $f$ 3-cycles of $\mathcal{S}$, looking at each gadget $X(v)$, either each of its in-plugs is

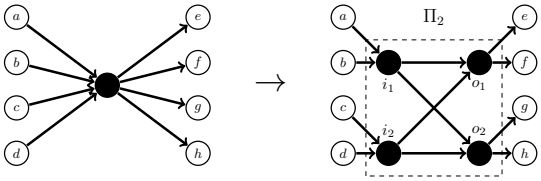



**Figure 7** Transformation of a vertex with in-degree and out-degree 4 into $\Pi_2$.

already connected to one of its out-plugs in $\mathbb{M}$ to form an $\alpha$-6-cycle (as described in the previous step), or there are the same number of free in- and out-plugs. Simply add to $\mathbb{M}_*$ one cross edge connecting each free in-plug $\dot{v}$ to a free out-plug $\hat{v}$. In the end of this procedure, $\mathbb{M}_*$ contains $p$ cross edges forming a perfect matching of in- and out-plugs of each $X(v)$. Each of these cross edges creates a $\mathbb{G}_i$-4-cycle. Therefore, in total there are $p|V|$ 4-cycles between $\mathbb{M}$ and the genomes $\mathbb{G}_1, \mathbb{G}_2, \ldots, \mathbb{G}_p$. Since $D$ has no 2-cycles, any cycle of $D$ that is not a directed 3-cycle has length greater than 3. This corresponds to a cycle of length greater than 6 in $\Theta(D, 1)$ and therefore does not affect the 6-score. Thus, the resulting score is $s_* = f + p|V|$.

Conversely, we will now show that if $s_* = f + p|V|$, we can derive an optimal solution $\mathcal{S}$ of MAX3CDEC in $D$ with $f$ directed 3-cycles.

Proposition 9 guarantees the existence of an optimal set $\mathbb{M}_*$, such that each edge in $\mathbb{M}_*$ is either a loop or a cross edge. If $\mathbb{M}_*$ is exclusively composed of loop edges, it induces $p|V|$ $\alpha$-2-cycles. In the other extreme, if $\mathbb{M}_*$ is exclusively composed of cross edges, it induces $p|V|$ $\mathbb{G}_{\hat{\alpha}}$-4-cycles. Note, however, that loop edges are incompatible, while cross edges are compatible with $\alpha$-6-cycles. Therefore, $\mathbb{M}_*$ will include all cross edges that also produce $\alpha$-6-cycles. Each $\alpha$-6-cycle with 3 cross edges corresponds to a directed 3-cycle in $D$ (Observation 8). If $f$ is the number of $\alpha$-6-cycles, then $\mathbb{M}_*$ will have at least the $3f$ cross edges that build these cycles, and each of these $3f$ cross edges also builds a $\mathbb{G}_{\hat{\alpha}}$-4-cycle. For the remaining $3|V| - 3f$ edges of $\mathbb{M}_*$, they are freely distributed among (compatible) cross and loop edges, and each one adds one cycle to the 6-score. Therefore $s_* = f + 3f + 3|V| - 3f = f + p|V|$ and there are $f$ directed 3-cycles in an optimal solution $\mathcal{S}$ of $D$. ◀

## 4.2 NP-completeness of the $(3, 12)$-Median problem

The proof of NP-completeness for three input genomes starts from MAX3CDEC using 4-regular digraphs. Given a 4-regular digraph $D$, we first apply a polynomial transformation that produces a 2-regular digraph $D_\Pi = (V', E')$ in which directed 3-cycles of $D$ correspond to directed 6-cycles of $D_\Pi$.

A *permutation network* is a directed graph $\Pi_d$ with $d$ input nodes $i_1, \ldots, i_d$ and $d$ output nodes $o_1, \ldots, o_d$. For every permutation $\pi$ of $\{1, \ldots, d\}$, there exist edge-disjoint paths $P_1, \ldots, P_d$ in $\Pi_d$ such that $P_j$ goes from $i_j$ to $o_{\pi(j)}$ [2]. Figure 7 illustrates the transformation of a vertex into $\Pi_2$. This transformation replaces one vertex with in- and out-degree 4 with four vertices with in- and out-degree 2 each.

Let $D$ be a 4-regular digraph with no 2-cycles. A 2-regular digraph $D_\Pi$ is obtained by replacing each vertex of $D$ with a $\Pi_2$. Since the digraph $\Pi_2$ has one edge between any input and output nodes, any path of length $m$ in $D$ has a corresponding path in $D_\Pi$ of length $2m$. Therefore, finding an optimal solution of MAX3CDEC for $D$ is equivalent to finding

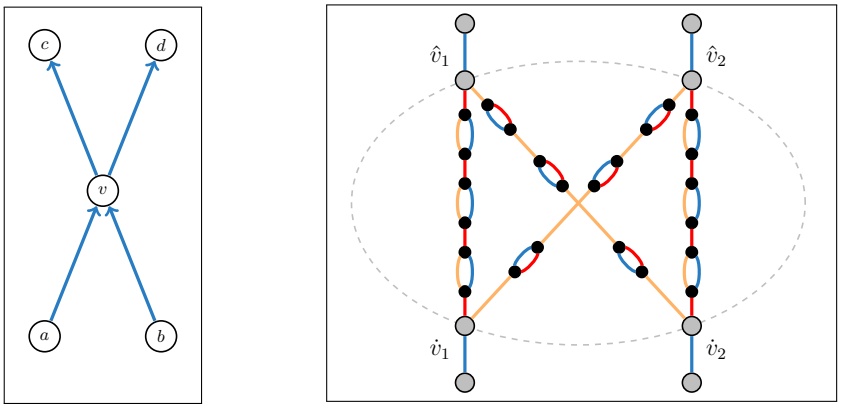
a maximum-cardinality collection of edge-disjoint directed 6-cycles in $D_\Pi$, and the latter problem is denoted MAX6CDEC. Note that, since $D$ is free of 2-cycles, the shortest cycle in $D_\Pi$ has length at least 6.

Here we have $k = 12$, therefore $q = \lfloor \frac{12}{4} \rfloor = 3$. Now let $\Theta(D_\Pi, 3)$ be obtained via Construction 4, as illustrated in Figure 8. Observation 8 guarantees that an oriented 6-cycle in $D_\Pi$ corresponds to a potential $\alpha$-12-cycle induced by set $\mathbb{M}_*$ in $\Theta(D_\Pi, 3)$. The reduction of $(3, 12)$-Median is analogous to the one of $(4, 6)$-Median (described in the previous subsection). Recall that, since for $q = 3$ the length of any $\mathbb{G}_i$-cycle is a multiple of 8 (Table 2), a $\mathbb{G}_i$-cycle can only contribute to the 6-score if it involves a single $\mathbb{G}_i$-bridge.

Here again we can focus on medians free of jump edges:

▶ **Proposition 11.** *There is a median $\mathbb{M}$ of $\Theta(D_\Pi, 3)$ that is of the form $\mathbb{M} = \mathbb{M}_Z \cup \mathbb{M}_*$ and is free of jump edges.*

**Proof.** Analogous to the proof of Proposition 9. ◀




**(a)** Vertex $v$ of $D_\Pi$. **(b)** Gadget $X^{2,3}(v)$ of $\Theta(D_\Pi, 3)$.

**Figure 8** **(a)** A vertex $v$ of $D_\Pi$ and its four adjacent vertices. **(b)** The gadget $X^{2,3}(v)$ of $\Theta(D_\Pi, 3)$ obtained from $v$. From the four vertices adjacent to $v$ in (a) we only show the respective plugs connected by $\alpha$-shortcuts to the plugs of $X^{2,3}(v)$.

These observations are sufficient to support our arguments. Therefore we have:

▶ **Theorem 12.** *The problem $(3, 12)$-Median is* NP*-complete.*

**Proof.** Take the subdivision factor $q = \lfloor \frac{12}{4} \rfloor = 3$. Note that here we use $p = 2$: Our $\Theta(D_\Pi, 3)$ is a graph obtained via Construction 4 on 2-regular graph $D_\Pi$ whose cycles have length at least 6. Proposition 6 guarantees that there is a median $\mathbb{M}$ for $\Theta(D_\Pi, 3)$ of the form $\mathbb{M} = \mathbb{M}_Z \cup \mathbb{M}_*$, where $\mathbb{M}_Z$ is fixed. Furthermore, the 12-score of $\mathbb{M}$ is $s_Z + s_*$, where $s_Z = qp^3|V'| = 24|V'|$.

The cross edges form two $\mathbb{G}_i$-8-cycles with $\mathbb{G}_i^3$-bridges in each $X^{2,3}$-gadget. Therefore we need to show the following equivalence: An optimal solution $\mathcal{S}$ of MAX6CDEC for digraph $D_\Pi = (V', E')$ has a partition into $f$ 6-cycles if and only if $s_* = f + 2|V'|$. The proof is analogous to the proof of Theorem 10 and shows that a set $\mathbb{M}_*$ including the $6f$ cross edges that build $f$ $\alpha$-12-cycles is optimal and allows the correspondence between the $f$ $\alpha$-12-cycles induced by $\mathbb{M}_*$ to the $f$ 6-cycles of $\mathcal{S}$. The remaining edges of $\mathbb{M}_*$ are freely distributed into (compatible) loop and cross edges. ◀

## 5 Larger values of $k$ with $\Theta'$-transformation and $\alpha$-brigdes

For generalizing our previous results to larger values of $k$, we follow the strategy used in the proof of NP-hardness for the marked directed Eulerian cycle decomposition problem [2]. In that construction, the edges are partitioned into three groups so that each directed 3-cycle contains exactly one edge from each group; then one group is marked. We use the analogous marked version of MAX3CDEC, in which every directed triangle contains exactly one marked edge and every cycle of length greater than 3 contains at least one marked edge.

The *marked maximum-cardinality edge-disjoint directed triangle decomposition*, denoted MARKED-MAX3CDEC, is described as follows. For $p \geq 3$, given a $p$-regular digraph with no 2-cycles and a set of marked edges, find a maximum-cardinality collection of edge-disjoint directed 3-cycles such that every selected triangle contains exactly one marked edge.

We adapt Construction 4 into a variation using marked edges:

▶ **Construction 13** ($\Theta'$-transformation). *Let $D = (V, E)$ be a $p$-regular digraph with $x$ marked edges, and let two integers $q$ and $r$ be such that $q \geq 1$ and $r \geq 1$. We first construct the graph $\Theta(D, q)$ and then we perform a $\ell^r$-subdivision of $\alpha$-shortcuts that correspond to marked edges of $D$, obtaining the $(p+1)$-colored graph $\Theta'(D, q, r)$. The detailed subdivision, after steps 1–3 from Construction 4, is as follows:*

4. *($r$-Subdivision of marked $\alpha$-shortcuts) For each $\alpha$-shortcut $\hat{v}\dot{w} \in \Theta(D, q)$, if the corresponding edge $vw$ is marked in $D$, subdivide $\hat{v}\dot{w}$ by the insertion of a $Z^r_{p+1}$-gadget. Once this is done for all $x$ marked edges of $D$, the resulting graph is $\Theta'(D, q, r)$.*

▶ **Proposition 14.** *If graph $\Theta'(D, q, r)$ is obtained via Construction 13 on $p$-regular digraph $D$ and integers $q \geq 1$ and $r \geq 1$, its $p+1$ colors define $p+1$ circular genomes $\mathbb{G}_1, \mathbb{G}_2, \ldots, \mathbb{G}_p$ and $\mathbb{G}_\alpha$ such that $\Theta'(D, q, r) = MBG(\mathbb{G}_1, \mathbb{G}_2, \ldots, \mathbb{G}_p, \mathbb{G}_\alpha)$.*

**Proof.** This was already true for $\Theta(D, q)$, and here we have only additional $Z$-pairs, and each vertex of such a pair is incident to exactly $p+1$ edges, each one of a distinct color $\mathbb{G}_1, \mathbb{G}_2, \ldots, \mathbb{G}_p$ or $\mathbb{G}_\alpha$. Therefore each of the $p+1$ colors still defines a perfect matching of the vertices of $\Theta'(D, q, r)$. ◀

We can still focus on a median $\mathbb{M}$ whose partitions are $\mathbb{M}_Z$ and $\mathbb{M}_*$, but note that here the partition $\mathbb{M}_Z$ also induces an $\alpha^r$-bridge for each of the $x$ $\ell^r$-subdivided $\alpha$-shortcuts in $\Theta'(D, q, r)$. Furthermore, the $Z$-score induced by $\mathbb{M}_Z$ needs to be updated accordingly.

▶ **Proposition 15.** *Let $k \geq 2$ and $\Theta'(D, q, r)$ be a graph obtained via Construction 13 on a $p$-regular digraph $D = (V, E)$ with $x$ marked edges and integers $q \geq 1$ and $r \geq 1$. There is a median $\mathbb{M}$ for $\Theta'(D, q, r)$ that can be partitioned into sets $\mathbb{M}_Z$ and $\mathbb{M}_*$, where $\mathbb{M}_Z$ has one edge connecting each $Z$-pair and $\mathbb{M}_*$ is a perfect matching of the plugs of $\Theta'(D, q, r)$. Furthermore, the $k$-score of $\mathbb{M}$ is $s_Z + s_*$, where the first term, also denoted $Z$-score, is a constant $s_Z = qp^3|V| + prx$, while the second term, $s_*$, is the maximum $k$-score attainable by connecting the plugs of $\Theta'(D, q, r)$.*

**Proof.** The $Z$-score in $\Theta(D, q)$ was $s_Z = qp^3|V|$. In $\Theta'(D, q, r)$, each of the $x$ $\ell^r$-subdivisions creates $r$ additional $Z$-pairs, each of which includes the endpoints of an $\widetilde{\mathbb{G}}_\alpha$-adjacency. Since $\mathbb{M}_Z$ induces $p$ 2-cycles with each of these new $\widetilde{\mathbb{G}}_\alpha$-adjacencies, the new $Z$-score is $s_Z = qp^3|V| + prx$. ◀

The $\alpha^r$-bridges have the goal of increasing the lengths of the $\alpha$-cycles with cross edges induced by $\mathbb{M}_*$.

▶ **Proposition 16.** *An oriented $t$-cycle $C$ in $p$-regular digraph $D$ with $s \geq 1$ marked edges implies that, for each edge $(v \to w)$ in $C$, if $(v \to w)$ is not marked, the graph $\Theta'(D, q, r)$ has an $\alpha$-shortcut connecting an out-plug $\hat{v}$ of $X^{p,q}(v)$ to an in-plug $\dot{w}$ of $X^{p,q}(w)$. Otherwise, if $(v \to w)$ is one of the $s$ marked edges, the graph $\Theta'(D, q, r)$ has an $\alpha^r$-bridge connecting $\hat{v}$ to $\dot{w}$. When $\mathbb{M}_*$ includes the cross edges chaining the endpoints of all $\alpha$-shortcuts and $\alpha^r$-bridges derived from the $t$-cycle $C$, then $\mathbb{M}_*$ induces an $\alpha$-cycle of length $2(t + rs)$ in $\Theta'(D, q, r)$.*

**Proof.** Each marked edge in $D$ corresponds to an $\alpha^r$-bridge in $\Theta'(D, q, r)$. Since an $\alpha^r$-bridge has length $2r + 1$, a $t$-cycle in $D_\Pi$ containing $s$ marked edges gives in $\Theta'(D, q, r)$ an $\alpha$-cycle whose length is $2t - s + 2rs + s = 2(t + rs)$ (assuming the corresponding median cross edges are in set $\mathbb{M}_*$). ◀

The idea is to set $r$ to a value such that the smallest $\alpha$-cycle with only median cross edges has length $k$. This can be achieved by increasing $r$ by one as $k$ increases by two, as shown in Table 3.

**Table 3** The values of the subdivision factor $r$ as $k$ grows and the corresponding length increases produced on $\alpha$-cycles.

| $r$ | 1 | 2 | 3 | 4 | $r \geq 5$ |
|---|---|---|---|---|---|
| length of $\alpha^r$-bridges | 3 | 5 | 7 | 9 | $2r + 1$ |
| increase on the length of $\alpha$-cycles produced by each $\alpha^r$-bridge | $+2$ | $+4$ | $+6$ | $+8$ | $+2r$ |
| $k$ for $\ell \geq 4$ and for $\ell = 3$ | 8 and 14 | 10 and 16 | 12 and 18 | 14 and 20 | $6 + 2r$ and $12 + 2r$ |

Observe that, in graph $\Theta'(D, q, r)$, a median loop edge can be parallel to an $\alpha$-shortcut, making an $\alpha$-2-cycle, or to an $\alpha^r$-bridge, making an $\alpha$-cycle of length $2(r + 1)$. By following the values of $r$ given in Table 3 it is clear that we always have $2(r + 1) < k$, therefore in this setting any $\alpha$-cycle with a loop edge still contributes to the $k$-score.

We are now ready to generalize the NP-completeness results from Subsection 4.1 and Subsection 4.2 to larger values of $k$.

## 5.1 NP-completeness of $(\ell, k)$-Median for $\ell \geq 4$ and even $k \geq 8$

For $p \geq 3$, let $D = (V, E)$ be a $p$-regular digraph with $x$ marked edges, such that each 3-cycle in $D$ has exactly one marked edge, and each larger cycle has at least one marked edge. Now take $q = \lfloor \frac{k}{4} \rfloor$ and $r = \frac{k-6}{2}$ for constructing the graph $\Theta'(D, q, r)$. We have $k = 6 + 2r$, and from Table 3 it is easy to see that each 3-cycle $C$ in $D$ (with a single marked edge) corresponds to a (potential) $\alpha$-$k$-cycle in $\Theta'(D, q, r)$. Moreover, from Table 2 it can be verified that only $\mathbb{G}_i$-cycles involving a single bridge contribute to the $k$-score.

▶ **Proposition 17.** *There is a median $\mathbb{M}$ of $\Theta'(D, q, r)$ that is of the form $\mathbb{M} = \mathbb{M}_Z \cup \mathbb{M}_*$ and is free of jump edges.*

**Proof.** Analogous to the proof of Proposition 9. ◀

With this framework, it is now straightforward to prove the following:

▶ **Theorem 18.** *Problem $(\ell, k)$-Median is* NP*-complete for $\ell \geq 4$ and even $k \geq 8$.*

**Proof.** For $p = \ell - 1$, let $D$ be a $p$-regular digraph free of 2-cycles and with $x$ marked edges such that each 3-cycle has a single marked edge and each larger cycle has at least one marked edge. Now take the subdivision factors $r = \frac{k-6}{2}$ and $q = \lfloor \frac{k}{4} \rfloor$, and let $\Theta'(D, q, r)$ be the graph obtained via Construction 13.

Proposition 15 guarantees that there is a median $\mathbb{M}$ for $\Theta'(D, q, r)$ of the form $\mathbb{M} = \mathbb{M}_Z \cup \mathbb{M}_*$, where $\mathbb{M}_Z$ is fixed. Furthermore, the $k$-score of $\mathbb{M}$ is $s_Z + s_*$, where $s_Z = qp^3|V'| + prx$.

The cross edges form p $\mathbb{G}_i$-cycles of length $2(q+1) < k$ with $\mathbb{G}_i^q$-bridges in each $X^{p,q}$-gadget. Therefore we need to show the following equivalence: An optimal solution $\mathcal{S}$ of MARKED-MAX3CDEC for digraph $D = (V, E)$ has a partition into $f$ 6-cycles if and only if $s_* = f + p|V|$. The proof is analogous to the proof of Theorem 10 and shows that a set $\mathbb{M}_*$ including the $3f$ cross edges that build $f$ $\alpha$-$k$-cycles is optimal and allows the correspondence between the $f$ $\alpha$-$k$-cycles induced by $\mathbb{M}_*$ to the $f$ 6-cycles of $\mathcal{S}$. The remaining edges of $\mathbb{M}_*$ are freely distributed into (compatible) loop and cross edges. ◀

## 5.2 NP-completeness of $(3, k)$-Median for even $k \geq 14$

Since the problem used for the proof of the NP-completeness of $(3, 12)$-Median is the same as the problem used for $(\ell, 6)$-Median, a similar adaptation of marked edges can be used.

Let $D = (V, E)$ be a 4-regular digraph with $x$ marked edges and no 2-cycles and let $D_\Pi = (V', E')$ be obtained by replacing each vertex of $D$ with a permutation network $\Pi_2$. Note that $D_\Pi$ has the "same" marked edges as $D$, therefore each 6-cycle of $D_\Pi$ has only one marked edge. Furthermore, each cycle of length greater than 6 has at least one marked edge.

Now let $q = \lfloor \frac{k}{4} \rfloor$, let $r = \frac{k-12}{2}$, and let $\Theta'(D_\Pi, q, r)$ be obtained via Construction 13. We have $k = 12 + 2r$, and from Table 3 it is easy to see that each 6-cycle $C$ in $D_\Pi$ (with a single marked edge) corresponds to a (potential) $\alpha$-$k$-cycle in $\Theta'(D_\Pi, q, r)$. Furthermore, only $\mathbb{G}_i$-cycles involving a single bridge contribute to the $k$-score (Table 2).

▶ **Proposition 19.** *There is a median $\mathbb{M}$ of $\Theta'(D_\Pi, q, r)$ that is of the form $\mathbb{M} = \mathbb{M}_Z \cup \mathbb{M}_*$ and is free of jump edges.*

**Proof.** Analogous to the proof of Proposition 9. ◀

These are all the relevant properties to derive our last result:

▶ **Theorem 20.** *Problem $(3, k)$-Median is* NP*-complete for $k \geq 14$.*

**Proof.** Let $D$ be a 4-regular digraph free of 2-cycles and with $x$ marked edges such that each 3-cycle has a single marked edge and each larger cycle has at least one marked edge. Then let $D_\Pi = (V', E')$ be the 2-regular digraph obtained by replacing each vertex of $D$ by a $\Pi_2$. Recall that each $m$-cycle in $D$ corresponds to a $2m$-cycle in $D_\Pi$ and note that $D_\Pi$ has the "same" marked edges as $D$, therefore each 6-cycle of $D_\Pi$ has a single marked edge and each larger cycle has at least one marked edge. Now take the subdivision factors $q = \lfloor \frac{k}{4} \rfloor$, and $r = \frac{k-12}{2}$, and let $\Theta'(D_\Pi, q, r)$ be the graph obtained via Construction 13.

Proposition 15 guarantees that there is a median $\mathbb{M}$ for $\Theta'(D_\Pi, q, r)$ of the form $\mathbb{M} = \mathbb{M}_Z \cup \mathbb{M}_*$, where $\mathbb{M}_Z$ is fixed. Furthermore, the $k$-score of $\mathbb{M}$ is $s_Z + s_*$, where $s_Z = qp^3|V'| + prx = 8q|V'| + 2rx$.

The cross edges form two $\mathbb{G}_i$-cycles of length $2(q+1) < k$ with $\mathbb{G}_i^q$-bridges in each $X^{2,q}$-gadget. Therefore we need to show the following equivalence: an optimal solution $\mathcal{S}$ of MARKED-MAX6CDEC for digraph $D_\Pi = (V', E')$ has a partition into $f$ 6-cycles if and only if $s_* = f + 2|V'|$. The proof is analogous to the proof of Theorem 10 and shows that a set $\mathbb{M}_*$ including the $6f$ cross edges that build $f$ $\alpha$-$k$-cycles is optimal and allows the correspondence between the $f$ $\alpha$-$k$-cycles induced by $\mathbb{M}_*$ and the $f$ 6-cycles of $\mathcal{S}$. The remaining edges of $\mathbb{M}_*$ are freely distributed into (compatible) loop and cross edges. ◀

## 6 Conclusion

The study of median problems under intermediate genome distances provides a natural way to refine the complexity landscape between two classical endpoints: the breakpoint median problem, corresponding to $\sigma_2$, which is polynomial-time solvable for multichromosomal circular genomes, and the DCJ median problem, corresponding to $\sigma_\infty$, which is NP-hard in the same setting. Within this landscape, the $(3, 4)$-Median problem plays a central role: it asks for a median of three genomes under the first intermediate distance after the breakpoint distance and has been highlighted in previous work as a fundamental case for understanding the transition from tractability to hardness.

Motivated by this question, we study the more general $(\ell, k)$-Median problem. Rather than addressing only one isolated case, this formulation allows us to map broader regions of the parameter space and to identify where NP-hardness already arises. We prove that $(4, 6)$-Median and $(3, 12)$-Median are NP-complete for circular genomes and extend these results to larger values of $\ell$ and $k$. Summarized in Table 1, our results provide a clearer picture of the transition from tractable to intractable cases for median problems with $\sigma_k$ distances. In particular, they delimit broad hardness regions around the central open case $(3, 4)$-Median and show that increasing either the number of input genomes or the distance parameter suffices to establish NP-completeness across large portions of the $(\ell, k)$ parameter space.

From a biological perspective, the restriction to short cycles can be viewed as a conservative way of emphasizing local, more reliable rearrangement signals, since long components in breakpoint graphs are often harder to interpret and may reflect breakpoint reuse, noise, or loss of historical signal [11]. Therefore, our work may lead to better rearrangement medians, computable similarly fast as the breakpoint median, but not as restricted.

## References


1 Amihood Amir, Tzvika Hartman, Oren Kapah, Avivit Levy, and Ely Porat. On the cost of interchange rearrangement in strings. In *Proceedings of ESA 2007*, volume 4698 of *LNCS*, pages 99–110, 2007. doi:10.1007/978-3-540-75520-3_11.

2 Martin Bader. The transposition median problem is NP-complete. *Theoretical Computer Science*, 412:1099–1110, 2011. doi:10.1016/j.tcs.2010.12.009.

3 Jørgen Bang-Jensen and Gregory Z. Gutin. *Digraphs: theory, algorithms and applications*. Springer London, 2009. doi:10.1007/978-1-84800-998-1.

4 Anne Bergeron, Julia Mixtacki, and Jens Stoye. A unifying view of genome rearrangements. In *Proceedings of WABI 2006*, volume 4175 of *LNBI*, pages 163–173, 2006. doi:10.1007/11851561_16.

5 Marilia D. V. Braga, Leonie R. Brockmann, Katharina Klerx, and Jens Stoye. Investigating the complexity of the double distance problems. *Algorithms for Molecular Biology*, 19:1, 2024. doi:10.1186/s13015-023-00246-y.

6 Cedric Chauve. Personal communication at Dagstuhl Seminar no. 18451 (Genomics, Pattern Avoidance, and Statistical Mechanics), 2018.

7 Luís Cunha, Thiago Lopes, Uéverton Souza, Leonard Bohnenkämper, Marilia D. V. Braga, and Jens Stoye. Closing the complexity gap of the double distance problem. *IEEE Transactions on Computational Biology and Bioinformatics*, 22:2194–2205, 2025. doi:10.1109/TCBBIO.2025.3590588.

8 Luís Cunha, Ignasi Sau, and Uéverton Souza. On the parameterized complexity of the median and closest problems under some permutation metrics. *Algorithms for Molecular Biology*, 19:24, 2024. doi:10.1186/s13015-024-00269-z.

**9** Anna E. Dewar, Laurence J. Belcher, and Stuart A. West. A phylogenetic approach to comparative genomics. *Nature Reviews Genetics*, 26:395–405, 2025. doi:10.1038/s41576-024-00803-0.

**10** Jakub Kováč. On the complexity of rearrangement problems under the breakpoint distance. *Journal of Computational Biology*, 21:1–15, 2014. doi:10.1089/cmb.2013.0004.

**11** David Sankoff. The signal in the genomes. *PLOS Computational Biology*, 2:e35, 2006. doi:10.1371/journal.pcbi.0020035.

**12** Helmuth O. M. Silva, Diego P. Rubert, Eloi Araujo, Eckhard Steffen, Daniel Doerr, and Fábio V. Martinez. Algorithms for the genome median under a restricted measure of rearrangement. *RAIRO Operations Research*, 57:1045–1058, 2023. doi:10.1051/ro/2023052.

**13** Eric Tannier, Chunfang Zheng, and David Sankoff. Multichromosomal median and halving problems under different genomic distances. *BMC Bioinformatics*, 10:120, 2009. doi:10.1186/1471-2105-10-120.

**14** Geoffrey A. Watterson, Warren J. Ewens, Thomas E. Hall, and Alexander Morgan. The chromosome inversion problem. *Journal of Theoretical Biology*, 99:1–7, 1982. doi:10.1016/0022-5193(82)90384-8.

**15** Andrew W. Xu. A fast and exact algorithm for the median of three problem: a graph decomposition approach. *Journal of Computational Biology*, 16:1369–1381, 2009. doi:10.1089/cmb.2009.0087.

**16** Sophia Yancopoulos, Oliver Attie, and Richard Friedberg. Efficient sorting of genomic permutations by translocation, inversion and block interchange. *Bioinformatics*, 21:3340–3346, 2005. doi:10.1093/bioinformatics/bti535.

# Turnpike with Uncertain Measurements: Triangle-Equality Integer Programming with a Deterministic Recovery Guarantee

**C. S. Elder**
Ray and Stephanie Lane Computational Biology Department,
Carnegie Mellon University, Pittsburgh, PA, USA

**Guillaume Marçais**
Ray and Stephanie Lane Computational Biology Department,
Carnegie Mellon University, Pittsburgh, PA, USA

**Carl Kingsford**
Ray and Stephanie Lane Computational Biology Department,
Carnegie Mellon University, Pittsburgh, PA, USA

## Abstract

We study the Turnpike problem with uncertain measurements: reconstructing a one-dimensional point set from an unlabeled multiset of pairwise distances under bounded noise and rounding. We give a combinatorial characterization of realizability via a multi-matching that labels interval indices by distinct distance values while satisfying all triangle equalities. This yields an integer linear program (ILP) based on the triangle equality whose constraint structure depends only on the two-partition set $\mathcal{P}_y = \{(r, s, t) : y_r + y_s = y_t,\ (r \neq s \text{ or } \mu_r \geq 2)\}$ and a natural linear-programming (LP) relaxation with $\{0, 1\}$-coefficient constraints. Integral solutions certify realizability and output an explicit assignment matrix, enabling a modular *assignment-first, regression-second* pipeline for downstream coordinate estimation. Under bounded noise followed by rounding, we prove deterministic separation conditions under which distinct rounded values do not collide and $\mathcal{P}_y$ is recovered exactly, so the ILP/LP receives the same combinatorial input as in the noiseless case. Experiments illustrate integrality behavior and degradation outside the provable regime.

**2012 ACM Subject Classification** Theory of computation → Design and analysis of algorithms; Mathematics of computing → Discrete optimization; Applied computing → Bioinformatics

**Keywords and phrases** Turnpike problem, partial digest, integer linear programming, LP relaxation, unlabeled distances, partitions, bounded noise

**Digital Object Identifier** 10.4230/LIPIcs.WABI.2026.20

**Related Version** *Full Version*: `https://arxiv.org/abs/2603.18283`

**Funding** This work was supported in part by the US National Science Foundation [III-2232121] and the US National Institutes of Health [R01HG012470].

## 1 Introduction

The *Turnpike problem* is a classical inverse problem: reconstruct a one-dimensional point set from an unlabeled multiset of pairwise distances. The origin story imagines a highway: we are given the multiset of all pairwise distances between exits, but the exit mile markers themselves have been lost. Can we recover a configuration of exits that is consistent with the observed distances?

Formally, let $x \in \mathbb{R}^n$ have strictly increasing coordinates. The *difference multiset* associated with $x$ is $\Delta x \coloneqq \{x_j - x_i : 1 \leq i < j \leq n\}$, which is a multiset containing $m \coloneqq \binom{n}{2}$ positive reals. An instance of Turnpike is a multiset $Y$ of $m$ positive reals, and the decision problem asks whether there exists an $x$ with $\Delta x = Y$.


© C. S. Elder, Guillaume Marçais, and Carl Kingsford;
licensed under Creative Commons License CC-BY 4.0



A multiset $Y$ is represented by a pair $(y, \mu)$ such that $y$ lists the values contained in $Y$ in strictly decreasing order $y_1 > \cdots > y_{m'} > 0$ where the multiplicity vector $\mu \in \mathbb{N}_{++}^{m'}$ records the number of occurrences, i.e., that a value $y_k$ is present $\mu_k \geq 1$ times. We say $Y$ is *realizable* when $Y = \Delta x$ for some $x \in \mathbb{R}^n$.

The Turnpike problem is equivalent to the Partial Digest problem from restriction site mapping in computational biology [1, 19] and is closely related to distance-geometry reconstruction tasks in crystallography, tandem mass spectrometry, and molecular error-correcting codes [7, 10, 12]. In restriction mapping, for example, a single-enzyme partial digestion produces fragments whose lengths coincide with all pairwise distances between restriction sites; the multiset $Y$ records fragment lengths, and a solution $x$ specifies restriction site locations along the strand (up to translation and reflection). The same combinatorial core occurs in tandem mass spectrometry, where the prefix masses of a peptide's backbone-fragmentation ions form a point set whose unlabeled pairwise differences are observed, and recovering the ordered residues from that multiset is again a Turnpike-type reconstruction [7, 15].

Exact recovery from error-free distances is neither known to be NP-complete nor to admit a polynomial-time algorithm [3]. A backtracking algorithm with worst-case exponential runtime but polynomial expected runtime under absolutely-continuous input models is known [18, 19]. Many additional approaches have been proposed, including polynomial factorization methods for exact instances, pruning strategies, and hardness and counterexample constructions for related variants [2, 3, 5, 8, 9, 13, 14, 16, 17, 20, 21].

#### Noisy Turnpike and measurement uncertainty

Experimental pipelines that produce Turnpike-like instances (partial digestion, tandem mass spectrometry, polymer codes) inevitably include measurement uncertainty and missing or duplicated observations. Allowing bounded additive or multiplicative errors makes the corresponding *bounded-error Turnpike* (formalized in Section 2.5) and the related circular Beltway problem strongly NP-hard [4, 6, 12], and backtracking approaches routinely explore exponentially many states even on non-adversarial inputs [4, 6, 19].

These noisy variants are strongly NP-hard, so we do not aim for an exact, polynomial-time algorithm. Instead, we focus on formulations whose integral solutions certify realizability and whose fractional solutions still retain structured combinatorial information, complementing the best-fit reconstruction solvers of prior work [6].

## 1.1 This work: a triangle-based assignment viewpoint

We develop a complementary viewpoint, treating unlabeled distances as a combinatorial *labeling* problem first and a regression problem second. The key object is a *ruler*: an indexed pairwise-difference table in which entry $(i, j)$ stores $x_j - x_i$. Unlike the multiset $\Delta x$, a ruler retains interval indices, and the identity $x_k - x_i = (x_j - x_i) + (x_k - x_j)$ becomes a family of linear consistency constraints (the *triangle equalities*). Realizability of $Y$ can therefore be rephrased as the existence of an assignment of distance values to interval indices that satisfies triangle equalities on every triple $i < j < k$.

This lens leads first to a natural mixed-integer linear programming (MILP) formulation for exact Turnpike, and then to a coordinate-free integer linear programming (ILP) formulation using only combinatorial information. A distinctive feature of this formulation is that it outputs an *assignment matrix* $\widehat{P}$ (a multi-matching between intervals and distinct distance values). A coordinate estimate $\widehat{x}$ can then be computed afterward by solving an independent

regression problem with any preferred numeric precision. This makes permutation metrics over $\widehat{P}$ the natural choice for primary evaluation, with coordinate-wise metrics treated as secondary.

In summary, we build a triangle-equality ILP for exact Turnpike, study its LP relaxation as a certificate/diagnostic layer, show that the formulation extends to the uncertain setting under deterministic separation conditions, and empirically validate that performance tracks this regime and degrades rapidly outside it.

### Main results and algorithmic viewpoint

Our results fall into four parts.

- **Triangle feasibility as realizability.** Realizability of $Y$ is equivalent to the existence of a multi-matching between interval indices and distinct distance values whose induced ruler satisfies all triangle equalities (Lemma 6), yielding a natural MILP (Proposition 8).
- **A coordinate-free triangle ILP/LP parameterized by two-partitions.** Eliminating coordinate variables and encoding triangle consistency through the two-partition set $\mathcal{P}_y = \{(r, s, t) : y_r + y_s = y_t,\ (r \neq s \text{ or } \mu_r \geq 2)\}$ yields the triangle ILP (Proposition 10) and its LP relaxation (System 11); $\mathcal{P}_y$ is built in $O(m'^2)$ time (Algorithm 1). With the basis reduction of Lemma 13, the formulation has $O(n^4)$ assignment and $O(n^6)$ triangle variables. The role is certificate generation, not best-fit regression.
- **Deterministic robustness under bounded noise and rounding.** Under measurement radius $\varepsilon$, rounding radius $R$, and the separation condition $\text{gap}_\star > 6(\varepsilon + R/2)$ (with $\text{sep}_y > 2(\varepsilon + R/2)$ guarding distinct-value collisions), the rounded data recover the exact two-partition set $\mathcal{P}_y$ (Theorem 15), so the triangle formulation operates on the noiseless combinatorial input.
- **Empirical evidence and assignment-first evaluation.** Experiments on synthetic and partial-digest instances track integrality and recovery primarily through assignment-level metrics (labelings and permutation distances) rather than coordinate regression. Recovery degrades rapidly in the $(\varepsilon, R)$ phase diagram (Figure 2) once Theorem 15's separation regime is violated.

### Limitations

Our deterministic guarantee assumes bounded noise followed by rounding and does not cover missing or duplicated distances. The separation condition is moreover stated through the noiseless quantities $\text{sep}_y$ and $\text{gap}_\star$, which are properties of the unknown ground truth rather than observable inputs; in practice we replace them by the data-driven calibration of Section 3.7. Although the formulations use polynomially many constraints and variables in $n$ and $m'$, the $O(n^6)$ triangle variables make full ILP/LP sweeps impractical beyond moderate sizes (we sweep up to $n = 30$), and we leave large-scale separation/enforcement schemes for future work. We also do not characterize when the LP relaxation is integral: integrality is reported empirically and a supporting theory is left open. Finally, our guarantee addresses additive noise with rounding; multiplicative and signal-dependent error models would require a different separation analysis.

## 2 Preliminaries

This section fixes notation and introduces the objects used throughout: distance multisets and the index sets they live over (Section 2.1), intervals and rulers (Section 2.2), the multi-matching viewpoint that converts realizability into an assignment-feasibility question

(Section 2.3), the corresponding MILP (Section 2.4), and the $(\varepsilon, R)$ noise-and-rounding model used later for deterministic recovery guarantees (Section 2.5). Symbols are introduced as they appear and are collected for reference in the notation table of Appendix C; the two-partition set $\mathcal{P}_y$, the central object of the triangle ILP/LP, is defined in Section 3.1.

## 2.1 Notation and multisets

We write $\mathbb{R}$, $\mathbb{Z}$, and $\mathbb{N}$ for the reals, integers, and naturals, respectively; subscripts indicate standard restrictions (e.g., $\mathbb{R}_+$ and $\mathbb{R}_{++}$ for nonnegative and positive reals). For an integer $n \geq 1$ we use $[n] = \{1, 2, \ldots, n\}$.

A numeric multiset is represented as an ordered pair $Y = (y, \mu)$ where $y \in \mathbb{R}^{m'}$ lists the distinct values in strictly decreasing order $y_1 > \cdots > y_{m'} > 0$, and $\mu \in \mathbb{N}_{++}^{m'}$ records the multiplicity of each value. The total cardinality (counting multiplicity) is $m = \sum_{r=1}^{m'} \mu_r$.

As in Section 1, for $x \in \mathbb{R}^n$ with strictly increasing entries we write $\Delta x = \{x_j - x_i : 1 \leq i < j \leq n\}$ for the multiset of pairwise distances. When $\Delta x = (y, \mu)$, the total multiplicity satisfies $m = \binom{n}{2}$.

▶ **Example 1.** For $x = (0, 2, 5, 9)$, the distances are $\{2, 3, 4, 5, 7, 9\}$, so $m' = m = 6$ and $\mu = \mathbf{1}$. For $x = (0, 2, 4, 6)$, the distances are $\{2, 2, 2, 4, 4, 6\}$, so $y = (6, 4, 2)$ and $\mu = (1, 2, 3)$.

## 2.2 Intervals, refinements, and rulers

We index unknown distances by their endpoints. A *monotone interval* is an ordered pair in $\mathcal{I}_n \coloneqq \{(i, j) \in [n] \times [n] : i < j\}$; a *monotone refinement* is an ordered triple in $\mathcal{T}_n \coloneqq \{(i, j, k) \in [n]^3 : i < j < k\}$. When unambiguous, we shorten $(i, j)$ to $ij$ and $(i, j, k)$ to $ijk$.

▶ **Definition 2.** *A* ruler *on $[n]$ is a matrix $\rho \in \mathbb{R}^{[n]\times[n]}$ for which there exists $x \in \mathbb{R}^n$ such that $\rho_{ij} = x_j - x_i$ for all $i, j \in [n]$. We call $x$ a* realization *of $\rho$. The ruler is* monotone *if $\rho_{ij} > 0$ for all $ij \in \mathcal{I}_n$.*

Every ruler satisfies skew-symmetry ($\rho_{ij} = -\rho_{ji}$) and the triangle equalities $\rho_{ik} = \rho_{ij} + \rho_{jk}$. The converse also holds.

▶ **Lemma 3** (Triangle equalities characterize rulers)**.** *A matrix $\rho \in \mathbb{R}^{[n]\times[n]}$ is a ruler if and only if it satisfies $\rho_{ik} = \rho_{ij} + \rho_{jk}$ for all $i, j, k \in [n]$.*

**Proof.** If $\rho_{ij} = x_j - x_i$ for some $x$, then $\rho_{ik} = (x_k - x_i) = (x_j - x_i) + (x_k - x_j) = \rho_{ij} + \rho_{jk}$.

Conversely, suppose $\rho$ satisfies the triangle equalities for all triples. Setting $i = j = k$ gives $\rho_{ii} = 2\rho_{ii}$, hence $\rho_{ii} = 0$. Then setting $k = i$ gives $0 = \rho_{ij} + \rho_{ji}$, so $\rho_{ji} = -\rho_{ij}$. Define $x_i \coloneqq \rho_{1i}$. Then for any $i, j$, $\rho_{ij} = \rho_{i1} + \rho_{1j} = -(\rho_{1i}) + \rho_{1j} = x_j - x_i$. ◀

▶ Remark 4. Lemma 3 shows that triangle equalities alone force the expected normalization $\rho_{ii} = 0$ and skew-symmetry. In particular, a monotone ruler determines a strictly increasing point set up to translation.

Because pairwise differences are translation invariant, realizability is unaffected by global shifts of $x$: replacing $x$ by $x + c\mathbf{1}$ leaves $\Delta x$ unchanged.

## 2.3 Multi-matchings and realizability

We now express realizability of $Y = (y, \mu)$ as the existence of a multi-matching between intervals and distinct distance values.

▶ **Definition 5.** *Let $\mathcal{X}$ be a finite set, and let $Y = (y, \mu)$ be a multiset with distinct values $y_1 > \cdots > y_{m'} > 0$. A* multi-matching *from $\mathcal{X}$ to $Y$ is a map $P : \mathcal{X} \to [m']$ such that $|P^{-1}(r)| = \mu_r$ for every $r \in [m']$.*

When $\mathcal{X} = \mathcal{I}_n$, the map $P$ assigns each interval $ij$ to one of the distinct distance values. We often identify $P$ with its one-hot encoding $(P^r_{ij})$ – the *assignment matrix* of $P$ – where the binary entry $P^r_{ij} \in \{0, 1\}$ indicates whether interval $ij$ is assigned value $y_r$.

Given a multi-matching $P : \mathcal{I}_n \to [m']$, define the induced *oriented difference matrix* $\rho^{(P)} \in \mathbb{R}^{[n]\times[n]}$ by

$$\rho^{(P)}_{ij} = \begin{cases} y_{P_{ij}}, & i < j, \\ 0, & i = j, \\ -y_{P_{ji}}, & i > j. \end{cases}$$

▶ **Lemma 6.** *A multiset $Y = (y, \mu)$ is realizable if and only if there exists a multi-matching $P : \mathcal{I}_n \to [m']$ with multiplicity $\mu$ such that $\rho^{(P)}$ is a monotone ruler.*

**Proof.** If $Y$ is realizable, choose $x \in \mathbb{R}^n$ with strictly increasing entries and $\Delta x = Y$. For each interval $ij \in \mathcal{I}_n$, pick the unique index $r$ with $x_j - x_i = y_r$ and set $P_{ij} := r$. The multiplicities in $\Delta x$ ensure $|P^{-1}(r)| = \mu_r$. By construction, $\rho^{(P)}_{ij} = x_j - x_i$, so $\rho^{(P)}$ is a monotone ruler.

Conversely, if $P$ is a multi-matching and $\rho^{(P)}$ is a monotone ruler, then by Lemma 3 there exists $x \in \mathbb{R}^n$ with $\rho^{(P)}_{ij} = x_j - x_i$. Monotonicity implies $x$ is strictly increasing up to translation. For $ij \in \mathcal{I}_n$ we have $x_j - x_i = \rho^{(P)}_{ij} = y_{P_{ij}}$, and the multi-matching multiplicities imply $\Delta x = Y$. ◀

▶ **Example 7** (One triangle constraint)**.** For $n = 4$, the refinement $(1, 2, 4)$ enforces $(x_2 - x_1) + (x_4 - x_2) = x_4 - x_1$. In the assignment view, this means the labels on intervals 12, 24, and 14 must form a two-partition $(r, s, t) \in \mathcal{P}_y$ with $y_r + y_s = y_t$.

## 2.4 An MILP formulation

Lemma 6 leads to a natural mixed-integer formulation that couples a discrete assignment with continuous coordinates.

▶ **Proposition 8.** *Let $Y = (y, \mu)$ have total multiplicity $m = \binom{n}{2}$. The following MILP is feasible if and only if $Y$ is realizable:*

$$x \in \mathbb{R}^n, \qquad P^r_{ij} \in \{0, 1\} \qquad \forall\, ij \in \mathcal{I}_n,\ r \in [m'],$$

$$\sum_{r=1}^{m'} P^r_{ij} = 1 \qquad \forall\, ij \in \mathcal{I}_n,$$

$$\sum_{ij \in \mathcal{I}_n} P^r_{ij} = \mu_r \qquad \forall\, r \in [m'],$$

$$x_j - x_i = \sum_{r=1}^{m'} y_r P^r_{ij} \qquad \forall\, ij \in \mathcal{I}_n.$$

*Optionally one may fix translation by adding $x_1 = 0$.*

**Proof.** If the system is feasible, define $P_{ij}$ as the unique $r$ with $P^r_{ij} = 1$. The last constraint gives $x_j - x_i = y_r$ with $r = P_{ij}$, so $\Delta x = Y$ by the multiplicity constraints.

Conversely, if $Y$ is realizable, choose $x$ and a multi-matching $P$ as in Lemma 6 and set $P^r_{ij} = \mathbf{1}[P_{ij} = r]$. Then all constraints hold. ◀

Proposition 8 captures realizability, but its LP relaxation depends on the numerical encoding of $y$ through the constraints $x_j - x_i = \sum_r y_r P^r_{ij}$, so near-duplicate distances, large offsets, and solver tolerances can degrade the combinatorial reliability of a numerically feasible assignment. Section 3 derives a triangle-based formulation whose constraint matrix is purely combinatorial once $\mathcal{P}_y$ is fixed; the extended-formulation baseline in Section 4.5 should be read against this contrast.

## 2.5 Noisy variants and the $(\varepsilon, R)$ model

We now formalize the bounded-error model and the rounding step used in our robustness guarantees.

In bounded-error noisy Turnpike, the input consists of an observed multiset $\widetilde{Y} = (\tilde{y}, \tilde{\mu})$ together with an error radius $\varepsilon \geq 0$. One asks whether there exist a point set $x \in \mathbb{R}^n$, a realizable multiset $Y = (y, \mu)$ with $\Delta x = Y$, and a matching between the elements of $Y$ and those of $\widetilde{Y}$ such that each matched pair differs by at most $\varepsilon$.

To reason about additive relations among noisy distances, we also introduce a rounding radius $R \geq 0$. For each distinct observed value $\tilde{y}_\ell$, define its rounded value $\hat{y}_\ell := \mathrm{round}_R(\tilde{y}_\ell) \in R\mathbb{Z} := \{Rk : k \in \mathbb{Z}\}$, the integer multiples of $R$, where $\mathrm{round}_R$ rounds $\tilde{y}_\ell$ to the nearest such multiple (ties may be broken arbitrarily). When $R = 0$ we interpret $\mathrm{round}_0$ as the identity map. After rounding, identical values are aggregated: we write $\widehat{Y} = (\hat{y}, \hat{\mu})$ for the resulting multiset of *distinct* rounded values and their multiplicities. Throughout, hats on $Y$ indicate rounded or processed input data, while hats on $P$ or $x$ indicate solver outputs or downstream estimates.

The pair $(\varepsilon, R)$ parameterizes our deterministic robustness statements: $\varepsilon$ bounds measurement uncertainty and $R$ bounds discretization error introduced by rounding. These parameters do not enter the MILP/ILP/LP systems directly; they govern the preprocessing $\widetilde{Y} \to \widehat{Y}$ and the recovery theorem (Section 3.6) that certifies when $\widehat{Y}$ delivers the same two-partition set as the noiseless instance.

# 3 A Triangle-Based ILP and LP Relaxation

The MILP of Proposition 8 couples discrete assignment variables and continuous coordinates through the equalities $x_j - x_i = \sum_r y_r P^r_{ij}$. In this section, we eliminate the coordinate variables entirely, enforcing ruler structure only via the additive relations among the distinct distance values – the *two-partitions* of $y$. This yields a triangle-based ILP, a natural LP relaxation, structural reductions, and a deterministic robustness guarantee under noise and rounding.

## 3.1 Two-partitions

Let $Y = (y, \mu)$ with distinct values $y_1 > \cdots > y_{m'} > 0$. A *two-partition* is an ordered triple $(r, s, t) \in [m']^3$ such that $y_r + y_s = y_t$, with the additional requirement that $\mu_r \geq 2$ when $r = s$, ensuring the multiset contains two copies of $y_r$ when it is used twice. We collect all such relations in $\mathcal{P}_y := \{(r, s, t) \in [m']^3 : y_r + y_s = y_t \text{ and } (r \neq s \text{ or } \mu_r \geq 2)\}$. This set can be enumerated in $O(m'^2)$ time by a two-pointer scan for each fixed target $t \in [m']$.

▶ **Example 9** (A two-partition set). Continuing the instance $x = (0, 2, 4, 6)$ from Section 2.1, the distance multiset is $Y = (y, \mu)$ with $y = (6, 4, 2)$ and $\mu = (1, 2, 3)$, so $m' = 3$. The only additive relations among the distinct values are $y_2 + y_3 = 4 + 2 = 6 = y_1$ and $2y_3 = 2 + 2 = 4 = y_2$; the second reuses the value $y_3$ and is admissible because $\mu_3 = 3 \geq 2$. Hence $\mathcal{P}_y = \{(2, 3, 1), (3, 2, 1), (3, 3, 2)\}$: a pair of intervals labeled 4 and 2 must compose to one labeled 6, and two intervals labeled 2 must compose to one labeled 4.

Running Algorithm 1 (deferred to Appendix B) for all $t \in [m']$ enumerates $\mathcal{P}_y$ in $O(m'^2)$ time, removing the extra $\log m'$ factor of a naïve per-pair binary search.

## 3.2 A triangle-based ILP

We now introduce an ILP over assignment variables $P$ and triangle variables $T$. Intuitively, $P$ specifies which distance value labels each interval $ij$, while $T$ enforces that every refinement $i < j < k$ respects the triangle equality through a compatible two-partition in $\mathcal{P}_y$.

For each interval $ij \in \mathcal{I}_n$ and each distance index $r \in [m']$, we use a binary variable $P_{ij}^r$. For each refinement $ijk \in \mathcal{T}_n$ and each two-partition $(r, s, t) \in \mathcal{P}_y$, we introduce a binary variable $T_{ijk}^{rst}$ indicating that the three intervals $(ij, jk, ik)$ are labeled by $(r, s, t)$.

▶ **Proposition 10** (Triangle-based ILP). *Let $Y = (y, \mu)$ and $\mathcal{P}_y$ be the respective two-partition set. The following ILP is feasible if and only if $Y$ is realizable:*

$$
\begin{aligned}
&P_{ij}^r \in \{0,1\} && \forall\, ij \in \mathcal{I}_n,\ r \in [m'],\\
&T_{ijk}^{rst} \in \{0,1\} && \forall\, ijk \in \mathcal{T}_n,\ (r,s,t) \in \mathcal{P}_y,\\
&\sum_{r=1}^{m'} P_{ij}^r = 1 && \forall\, ij \in \mathcal{I}_n,\\
&\sum_{ij \in \mathcal{I}_n} P_{ij}^r = \mu_r && \forall\, r \in [m'],\\
&\sum_{(r,s,t)\in\mathcal{P}_y} T_{ijk}^{rst} = 1 && \forall\, ijk \in \mathcal{T}_n,\\
&P_{ij}^r = \sum_{\substack{(r',s,t)\in\mathcal{P}_y\\ r'=r}} T_{ijk}^{r'st} && \forall\, ijk \in \mathcal{T}_n,\ \forall r \in [m'],\\
&P_{jk}^s = \sum_{\substack{(r,s',t)\in\mathcal{P}_y\\ s'=s}} T_{ijk}^{rs't} && \forall\, ijk \in \mathcal{T}_n,\ \forall s \in [m'],\\
&P_{ik}^t = \sum_{\substack{(r,s,t')\in\mathcal{P}_y\\ t'=t}} T_{ijk}^{rst'} && \forall\, ijk \in \mathcal{T}_n,\ \forall t \in [m'].
\end{aligned}
$$

**Proof.** The first two constraint families enforce that $P$ is a multi-matching between intervals and distinct distance values with multiplicities $\mu$. The remaining constraints enforce triangle consistency: for each refinement $i < j < k$, exactly one two-partition $(r, s, t) \in \mathcal{P}_y$ is selected, and the induced labels on $(ij, jk, ik)$ agree with $(r, s, t)$.

If $Y$ is realizable, choose $x$ with $\Delta x = Y$, and let $P$ be the induced interval labeling from Lemma 6. For each refinement $i < j < k$, the true distances satisfy $(x_j - x_i) + (x_k - x_j) = x_k - x_i$, so if $x_j - x_i = y_r$, $x_k - x_j = y_s$, and $x_k - x_i = y_t$, then $(r, s, t) \in \mathcal{P}_y$. Setting $T_{ijk}^{rst} = 1$ for these triples (and 0 otherwise) gives feasibility.

Conversely, suppose $(P, T)$ satisfies the ILP. By construction, $P$ defines a multi-matching with the correct multiplicities. The triangle constraints imply that for every refinement $i < j < k$, the three entries of the induced difference matrix $\rho^{(P)}$ satisfy the triangle equality. By Lemma 3, $\rho^{(P)}$ is a ruler, hence corresponds to a point set $x$ with $\Delta x = Y$. ◀

## 3.3 Triangle-based LP relaxation and interpretation

Relaxing integrality in Proposition 10 yields a linear system.

▶ **System 11** (Triangle-based LP relaxation)**.** The triangle-based LP relaxation consists of variables $P_{ij}^r \in [0,1]$ for $ij \in \mathcal{I}_n$ and $r \in [m']$, and variables $T_{ijk}^{rst} \in [0,1]$ for $ijk \in \mathcal{T}_n$ and $(r,s,t) \in \mathcal{P}_y$, together with the same linear equalities as in Proposition 10.

An integral solution $P$ to System 11 is feasible for the ILP and therefore certifies realizability of $Y$; a fractional solution may indicate non-realizability or inexactness of the relaxation. Either way, the expected differences $\rho_{ij}^{(P)} \coloneqq \sum_{r=1}^{m'} y_r P_{ij}^r$ (extended skew-symmetrically) yield a valid ruler (Corollary 12), making fractional solutions structured starting points for downstream regression schemes for Beltway and Turnpike [6].

## 3.4 Relation to exact best-fit formulations

Previous optimization-based approaches to noisy Turnpike-family problems are *best-fit* MILP formulations that, once a residual norm or support-function loss is fixed, return a globally optimal reconstruction under that objective (often via compact sorting or permutahedral extended formulations [6, 11]). The triangle ILP/LP plays a different role: it is a *certificate* formulation for combinatorial realizability, requiring no coordinate loss. An integral feasible solution certifies that the observed multiset admits an interval-to-distance assignment satisfying every triangle equality; when the recovered two-partition set is distorted by noise, the formulation remains a useful diagnostic rather than a best-fit solver. Consequently, in the experiments below, coordinate error is reported only after the assignment layer produces a hard assignment, and LP fractionality is treated as evidence about the relaxation rather than as a stand-alone reconstruction objective.

## 3.5 Structural consequences and formulation reductions

This subsection records direct consequences of the triangle formulation; none introduces coordinate variables.

### Fractional solutions induce consistent rulers

For $(P, T)$ feasible in System 11, define $\rho_{ij}^{(P)} \coloneqq \sum_{r=1}^{m'} y_r P_{ij}^r$ for $ij \in \mathcal{I}_n$, $\rho_{ii}^{(P)} = 0$, and $\rho_{ji}^{(P)} = -\rho_{ij}^{(P)}$.

▶ **Corollary 12** (Feasible LP solutions induce a ruler)**.** *If $(P, T)$ satisfies System 11, then $\rho_{ik}^{(P)} = \rho_{ij}^{(P)} + \rho_{jk}^{(P)}$ for all $i < j < k$. Consequently, there exists $x \in \mathbb{R}^n$, unique up to translation, such that $\rho_{ij}^{(P)} = x_j - x_i$ for all $i, j \in [n]$.*

The marginalization constraints of System 11 project $T_{ijk}^{rst}$ to $P_{ij}^r$, $P_{jk}^s$, $P_{ik}^t$, so substituting $y_t = y_r + y_s$ on each two-partition gives $\rho_{ik}^{(P)} = \rho_{ij}^{(P)} + \rho_{jk}^{(P)}$ directly; Lemma 3 then identifies $\rho^{(P)}$ as a ruler. Full algebraic expansion is in Appendix A.1.

Corollary 12 is an algebraic statement about a fixed two-partition input $\mathcal{P}_y$. It does not by itself solve a noisy instance. For uncertain data, the parameters $(\varepsilon, R)$ enter earlier, through the construction of the rounded representatives $\widehat{Y}$ and the approximate two-partition set. Theorem 15 gives a sufficient condition under which that combinatorial input is the same as in the exact instance.

### A triangle basis

The full family of refinement-indexed triangle constraints is redundant.

▶ **Lemma 13** (Triangle-basis reduction). *Let $\rho \in \mathbb{R}^{[n]\times[n]}$ satisfy $\rho_{ii} = 0$ and $\rho_{ji} = -\rho_{ij}$. The following are equivalent:*

1. *$\rho_{ik} = \rho_{ij} + \rho_{jk}$ for all $i < j < k$,*
2. *$\rho_{1k} = \rho_{1j} + \rho_{jk}$ for all $1 < j < k$,*
3. *there exists $x \in \mathbb{R}^n$ such that $\rho_{ij} = x_j - x_i$ for all $i, j \in [n]$.*

The implications (1) ⇒ (2) and (3) ⇒ (1) are immediate; (2) ⇒ (3) follows by setting $x_1 = 0$, $x_k \coloneqq \rho_{1k}$, and applying (2) to triples of the form $(1, i, j)$ (Appendix A.2).

Lemma 13 implies that the triangle-consistency constraints in Proposition 10 and System 11 may be imposed only for refinements of the form $(1, j, k)$ with $1 < j < k$. This reduces the number of refinement-indexed constraint blocks from $\binom{n}{3}$ to $\binom{n-1}{2}$ and the number of triangle variables accordingly.

### Containment-based pruning

Define a containment order on intervals by setting $ij \preceq st$ if $s \le i < j \le t$. Under strict containment $ij \prec st$, every strictly increasing realization satisfies $x_j - x_i < x_t - x_s$.

Let $m = \binom{n}{2}$ and let $d_1 \ge \cdots \ge d_m$ be the multiset $\Delta x$ written as a nonincreasing list (including multiplicities). For an interval $ij \in \mathcal{I}_n$, define $\mathrm{rank}_x(ij) \in [m]$ as any index such that $d_{\mathrm{rank}_x(ij)} = x_j - x_i$.

▶ **Lemma 14** (Containment-based rank bounds). *For every strictly increasing $x \in \mathbb{R}^n$ and every $ij \in \mathcal{I}_n$, $i(n-j+1) \le \mathrm{rank}_x(ij) \le m - \binom{j-i+1}{2} + 1$.*

The lower bound counts the $i(n-j+1) - 1$ strict superintervals of $ij$ (each with strictly larger distance); the upper bound counts the $\binom{j-i+1}{2} - 1$ strict subintervals (each with strictly smaller distance). See Appendix A.3.

Writing $M_r \coloneqq \sum_{s=1}^{r} \mu_s$, the value $y_r$ occupies rank block $[M_{r-1}+1, M_r]$ in the sorted expansion of $Y$, so $P_{ij}^r$ may be fixed to 0 whenever this block is disjoint from the admissible rank range of Lemma 14. This depends only on $(n, \mu)$ and serves as a presolve step in any of the formulations above.

Together these steps form an assignment-first pipeline: enumerate $\mathcal{P}_y$, solve the triangle ILP (Proposition 10) or its LP relaxation (System 11), certify realizability when the assignment $\widehat{P}$ is integral (otherwise read $\widehat{P}$ as a diagnostic), and optionally extract coordinates $\widehat{x}$ from the induced ruler (Corollary 12) or by downstream regression (Section 4.2). Algorithm 2 in Appendix B states this explicitly.

## 3.6 Two-partition recovery under noise and rounding

We now return to the bounded-error model from Section 2.5 to show that, under a simple separation condition, the two-partition structure of a noiseless instance can be recovered from a noisy observation of that instance by a nearest-neighbor rounding procedure.

Let $Y = (y, \mu)$ be the ground-truth multiset with distinct values $y_1 > \cdots > y_{m'} > 0$. We use two separation parameters. The first prevents distinct rounded representatives from colliding:

$$\mathrm{sep}_y \coloneqq \min_{a \neq b} |y_a - y_b| ,$$

with the convention $\mathrm{sep}_y = +\infty$ when $m' = 1$. For each index $t \in [m']$, define the *additive gap at* $t$ by

$$\mathrm{gap}_t \coloneqq \min_{\substack{r,s \in [m'] \\ y_r + y_s \neq y_t}} |y_r + y_s - y_t| , \qquad \mathrm{gap}_\star \coloneqq \min_{t \in [m']} \mathrm{gap}_t .$$

We use the convention $\mathrm{gap}_t = +\infty$ when no invalid pair exists.

We observe noisy distances $\widetilde{Y}$ and form rounded representatives by applying $\mathrm{round}_R$ to each observed value, yielding $\widehat{Y} = (\hat{y}, \hat{\mu})$. For each $y_k$, the corresponding rounded representative $\hat{y}_k$ satisfies $|\hat{y}_k - y_k| \leq \varepsilon + R/2$, since rounding to the nearest multiple of $R$ incurs a per-value error of at most $R/2$. For use below, the *approximate* two-partition test for threshold $\tau > 0$ is $|\hat{y}_r + \hat{y}_s - \hat{y}_t| \leq \tau$.

▶ **Theorem 15** (Two-partition recovery under bounded noise)**.** *Suppose the noise and rounding radii satisfy*

$$2(\varepsilon + R/2) < \mathrm{sep}_y \qquad \textit{and} \qquad 6(\varepsilon + R/2) < \mathrm{gap}_\star .$$

*Then rounding and aggregation preserve the distinct representatives of $Y$. With $\tau = \mathrm{gap}_\star/2$, we have for every triple $(r, s, t) \in [m']^3$ that $y_r + y_s = y_t \iff |\hat{y}_r + \hat{y}_s - \hat{y}_t| \leq \tau$. Because the same no-collision condition preserves multiplicities, the exact multiplicity-qualified two-partition set $\mathcal{P}_y$ is recovered as*

$$\{(r, s, t) : |\hat{y}_r + \hat{y}_s - \hat{y}_t| \leq \tau,\ r \neq s \ \textit{or}\ \hat{\mu}_r \geq 2\}.$$

**Proof.** For every distinct pair $a \neq b$, the matched rounded representatives satisfy

$$|\hat{y}_a - \hat{y}_b| \geq |y_a - y_b| - 2(\varepsilon + R/2) \geq \mathrm{sep}_y - 2(\varepsilon + R/2) > 0.$$

Thus aggregation after rounding does not merge two distinct true values, so the rounded representatives can be indexed by the same labels as $y$.

If $y_r + y_s = y_t$, write $\hat{y}_k = y_k + \xi_k + \delta_k$ with $|\xi_k| \leq \varepsilon$ (measurement error) and $|\delta_k| \leq R/2$ (rounding error). Then $\hat{y}_r + \hat{y}_s - \hat{y}_t = (\xi_r + \xi_s - \xi_t) + (\delta_r + \delta_s - \delta_t)$, which means $|\hat{y}_r + \hat{y}_s - \hat{y}_t| \leq 3(\varepsilon + R/2) < \frac{1}{2}\mathrm{gap}_\star = \tau$.

If $y_r + y_s \neq y_t$, then $|y_r + y_s - y_t| \geq \mathrm{gap}_\star$ by definition. The same decomposition gives $\hat{y}_r + \hat{y}_s - \hat{y}_t = (y_r + y_s - y_t) + (\xi_r + \xi_s - \xi_t) + (\delta_r + \delta_s - \delta_t)$, and the reverse triangle inequality yields $|\hat{y}_r + \hat{y}_s - \hat{y}_t| \geq \mathrm{gap}_\star - 3(\varepsilon + R/2) > \frac{1}{2}\mathrm{gap}_\star = \tau$. ◀

The condition in Theorem 15 is deterministic and worst-case: it is phrased in terms of a distinct-value separation parameter $\mathrm{sep}_y$, an additive-relation separation parameter $\mathrm{gap}_\star$, and a noise bound $\varepsilon$. Its role here is to identify a regime in which the combinatorial input $\mathcal{P}_y$ to the triangle-based ILP/LP can be recovered exactly from noisy measurements after rounding. Outside this regime, the recovered two-partition set is not guaranteed to contain the ground truth as a subset; while the formulation remains well-defined in such cases, the output should be interpreted as a triangle-equality–consistent diagnostic/heuristic rather than as a realizability certificate.

## 3.7 Practical calibration and finite precision

The proof of Theorem 15 yields a direct data-driven analog: define $\rho_{abc}^{(R)} := \left|\hat{y}_a^{(R)} + \hat{y}_b^{(R)} - \hat{y}_c^{(R)}\right|$ and take $\widehat{\mathcal{P}}(\varepsilon; R) := \{(a,b,c) : \rho_{abc}^{(R)} \le 3(\varepsilon + R/2)\}$ (with the usual multiplicity qualifier). For fixed $R$, this set is piecewise constant in $\varepsilon$ and changes only at the finitely many critical values $\psi_{abc}^{(R)} := \max\{0, \rho_{abc}^{(R)}/3 - R/2\}$, so a calibration loop using triangle-ILP feasibility as predicate can binary-search over these sorted radii instead of tuning $\varepsilon$ manually. The same separation bound also covers finite-precision arithmetic: if every distinct value is computed to within $\eta$ and $6\eta < \mathrm{gap}_\star$, the combinatorial two-partition input is preserved exactly. Appendix A.4 carries the full derivation.

# 4 Experiments

The experiments below are intended to (i) illustrate empirical integrality behavior of the LP relaxation on computationally-feasible problem sizes and (ii) demonstrate recovery accuracy as measured by assignment-first metrics enabled by the ILP/LP output.

## 4.1 Setup

We evaluate the triangle ILP (Proposition 10) and its LP relaxation (System 11) on noisy synthetic instances under the $(\varepsilon, R)$ model and on partial-digest benchmarks with error magnitudes matching practice [1]. Synthetic instances supply known ground-truth assignments and controlled $\varepsilon, R$; partial-digest benchmarks stress-test the same pipeline on application-shaped instances with fragment errors and repeated distances.

Synthetic instances use up to $n = 30$ ($m = \binom{30}{2} = 435$ distances); partial-digest benchmarks use up to 100 fragments. The cutoff at $n = 30$ is the largest size for full triangle-variable sweeps; larger experiments would use the basis reduction and separation-style enforcement of Section 3.5. Given $Y = (y, \mu)$, the ILP/LP layer returns the assignment matrix $\widehat{P}$; whenever $\widehat{P}$ is integral we also compute a coordinate estimate $\widehat{x}$ by least-squares regression (Section 4.2) as a secondary sanity check. For the LP relaxation we record the integrality score of $\widehat{P}$ (an integral solution certifies realizability; a fractional one exposes soft assignment structure); for the ILP we record feasibility and recovery metrics.

### Implementation

All instances were modeled and solved in `gurobipy` (Gurobi Optimizer 13.0 Python API) at default settings unless noted, on `Apple M4 Pro / 24 GB RAM / macOS Sequoia 15.6.1`.

### On simulated benchmarks

We evaluate primarily on simulated and application-shaped partial-digest instances because simulation supplies exact ground-truth assignments $P^\star$ and independent control of the noise and rounding radii $(\varepsilon, R)$, both of which the assignment-first metrics of Section 4.2 require and which public partial-digest collections do not provide in labeled form; the same convention is followed by prior partial-digest reconstruction studies [4, 19]. Results on measured data are reported in Appendix D. The error magnitudes are matched to reported practice ($\varepsilon = 5$ bases [1]).

## 4.2 Metrics

Because the assignment layer returns a discrete labeling, we evaluate recovery primarily using permutation and labeling metrics on $\widehat{P}$. Coordinate errors for $\widehat{x}$ are treated as secondary. When duplicate distances create several equivalent ground-truth assignments, labels are compared up to the corresponding tie classes rather than to an arbitrary ordering of identical values.

### Interval labeling error

Let $P^\star$ denote a ground-truth assignment for a realizable instance. The labeling error is

$$\mathrm{err}_{\mathrm{lab}}(\widehat{P}, P^\star) \coloneqq \frac{1}{|\mathcal{I}_n|} \sum_{ij \in \mathcal{I}_n} \mathbf{1}\left[\widehat{P}_{ij} \neq P^\star_{ij}\right].$$

### Permutation distance

Let $\pi^\star$ be the ground-truth permutation mapping unlabeled distances to interval indices, and let $\widehat{\pi}$ be the permutation induced by $\widehat{P}$. We measure the difference between each $\widehat{\pi}$ and $\pi^\star$ through the normalized Kendall–$\tau$ distance, which we denote as $\mathrm{dist}_{\mathrm{perm}}(\widehat{\pi}, \pi^\star) \in [0, 1]$.

### Coordinate error

For each $\widehat{P}$, we compute the associated $\widehat{x}$ with least-squares regression as

$$\widehat{x} \in \arg\min_{x \in \mathbb{R}^n} \sum_{ij \in \mathcal{I}_n} \left(x_j - x_i - \rho_{ij}^{(\widehat{P})}\right)^2,$$

with translation fixed as $x_1 = 0$. Before comparison, $x^\star$ is translated so that $x^\star_1 = 0$. We report the mean absolute error (MAE),

$$\mathrm{MAE}(\widehat{x}, x^\star) \coloneqq \frac{1}{n} \sum_{i=1}^{n} \left|\widehat{x}_i - x^\star_i\right|,$$

after aligning $\widehat{x}$ to $x^\star$ up to reflection (i.e., choosing the better of $\widehat{x}$ and its reflected copy under this metric).

### LP integrality score

For the LP relaxation, we quantify integrality via

$$\mathrm{int}(\widehat{P}) \coloneqq \frac{1}{|\mathcal{I}_n|} \sum_{ij \in \mathcal{I}_n} \max_{r \in [m']} \widehat{P}^r_{ij},$$

which equals 1 if and only if $\widehat{P}$ is an integral multi-matching.

## 4.3 Synthetic exact instances

We generate exact instances by sampling i.i.d. coordinates $z_1, \ldots, z_n$ from three distributions (uniform on $[0, 1]$, standard normal, and Cauchy) and sorting to obtain $x^\star$; for linear instances we form $Y = \Delta x^\star$, and for circular instances we use the corresponding Beltway multiset of shortest-arc distances. Figure 1 summarizes the integrality score of the triangle LP relaxation as a function of $n$ for linear Turnpike instances (left) and circular Beltway instances (right).

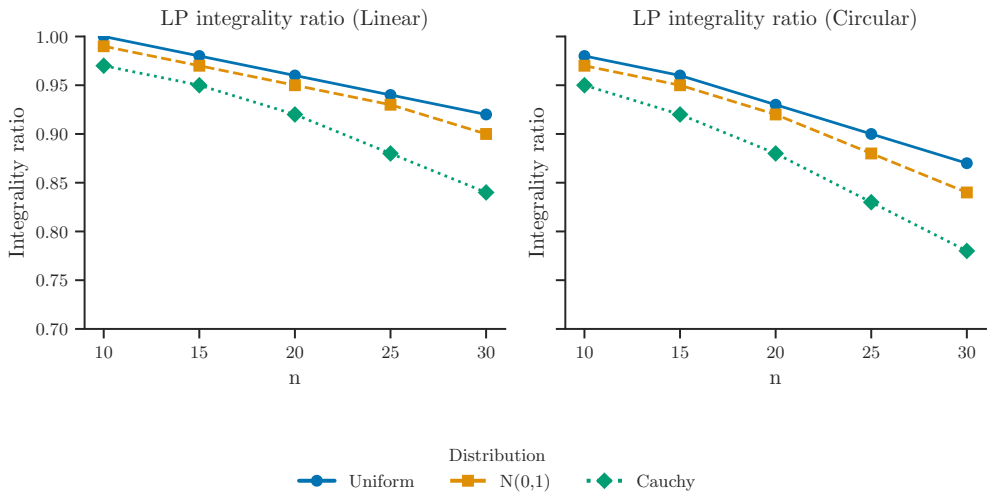



**Figure 1** Synthetic exact instances. LP integrality score on linear Turnpike instances (left) and circular Beltway instances (right), stratified by the distribution of $x^\star$. The relaxation is integral on most small instances, with the score declining as $n$ grows.

## 4.4 Phase transitions under the $(\varepsilon, R)$ model

We now test the deterministic regime established in Theorem 15. The exact instances are drawn from the same synthetic family as in Section 4.3 (i.i.d. coordinates from one of the three distributions, sorted to give $x^\star$); for each instance we add bounded noise of radius $\varepsilon$ to the distances and then round to a grid of spacing $R$, yielding $\widehat{Y} = (\hat{y}, \hat{\mu})$ (here hats denote rounded input values; $\widehat{P}$ denotes the solver output). From $\widehat{Y}$ we enumerate an *approximate* two-partition set via the data-driven tolerance test

$$|\hat{y}_r + \hat{y}_s - \hat{y}_t| \le 3(\varepsilon + R/2),$$

as in Section 3.7; we then solve the triangle ILP/LP using this approximate $\widehat{\mathcal{P}}(\varepsilon; R)$.

Figure 2 reports three heatmaps over a grid of $(\varepsilon, R)$ values: the fraction of instances whose exact two-partition set is recovered (top), the rate of spurious two-partitions (bottom left), and the rate of missing true two-partitions (bottom right). The dashed line indicates the sufficient deterministic condition of Theorem 15.

**Takeaway.** The recovered two-partition structure is stable in a region consistent with the deterministic bound in Theorem 15. As $\varepsilon$ and $R$ increase, $\widehat{\mathcal{P}}(\varepsilon; R)$ accrues false positives and false negatives, so the triangle formulation is fed a distorted two-partition input outside the provable regime.

## 4.5 Comparison with regression-based baselines

We compare our formulation with regression-based pipelines using baselines established in previous work [6]. The goal is not to optimize a coordinate loss directly, but to highlight that the triangle ILP/LP returns an assignment matrix $\widehat{P}$ that can be evaluated without committing to a particular numeric norm or optimizer.

For this comparison we use the synthetic linear Turnpike instances of Section 4.3. Figure 3 reports median interval labeling error, permutation distance, and coordinate MAE as a function of $n$ for (i) the triangle-equality ILP, (ii) the triangle-equality LP (rounded to a hard

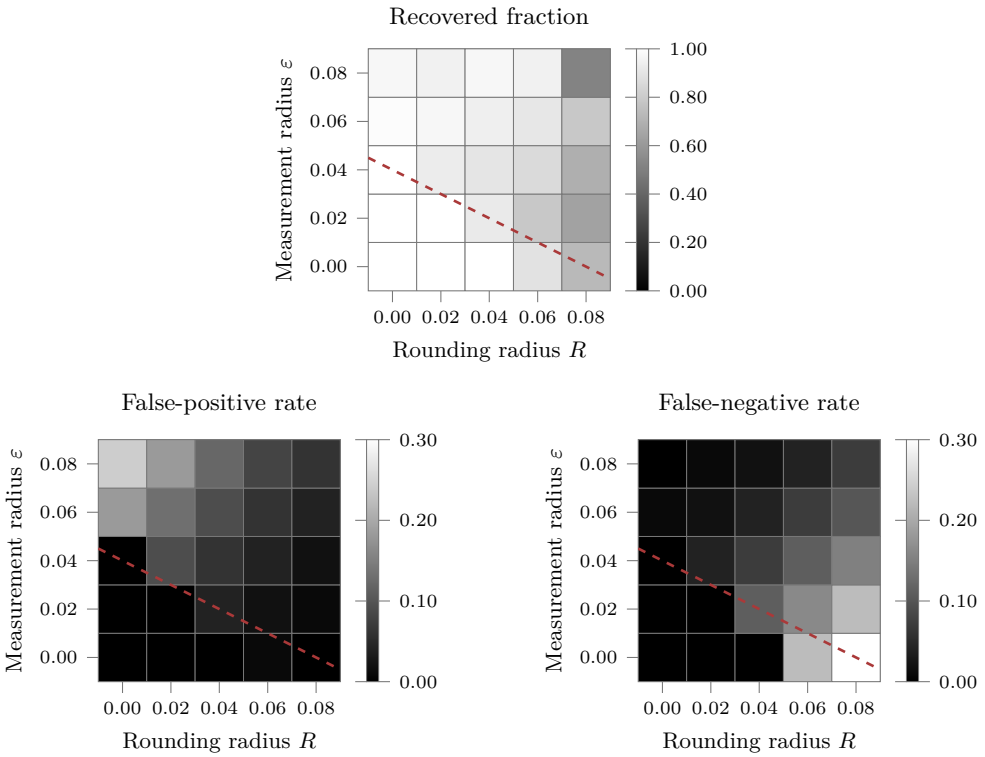



**Figure 2** Noisy $(\varepsilon, R)$ phase diagram for two-partition recovery. Top: recovery rate; bottom left: false-positive rate (spurious two-partitions); bottom right: false-negative rate (missing true two-partitions). The dashed line is the sufficient condition of Theorem 15; recovery is near-perfect below it and degrades above it.

assignment by argmax and then evaluated only as a heuristic assignment), (iii) a majorization–minimization baseline (MM), (iv) a gradient-descent baseline (GD), and (v) a sorting-network extended-formulation ILP baseline (Ext-ILP). Here Ext-ILP denotes the implementation used by Elder et al. [6], which is equivalent to the coordinate-coupled MILP of Proposition 8 and is based on Goemans's compact extended formulation for the permutahedron [11].

**Takeaway.** On this benchmark suite, the assignment-first formulation directly targets the combinatorial quantities in the labeling-error and permutation-distance panels, and its median labeling and permutation errors are lower than those of the coordinate-loss baselines shown in Figure 3. This advantage is structural rather than incidental: the triangle ILP/LP optimizes the interval-to-value assignment directly, so it does not inherit the labeling error the baselines incur when labels are read off a fitted coordinate vector. We do not claim a coordinate-accuracy or runtime advantage in the coordinate-MAE panel, where a best-fit objective is the natural target.

## 4.6 Partial digest reconstruction

We sample partial-digest instances over linear and circular genomes (the Beltway variant), observing each up to bounded error with $\varepsilon = 5$ (i.e., up to 5 additional or missing bases on each fragment), matching error magnitudes observed in practice [1]. Figure 4 reports normalized labeling error and fragment recovery rate as a function of the number of fragments for linear

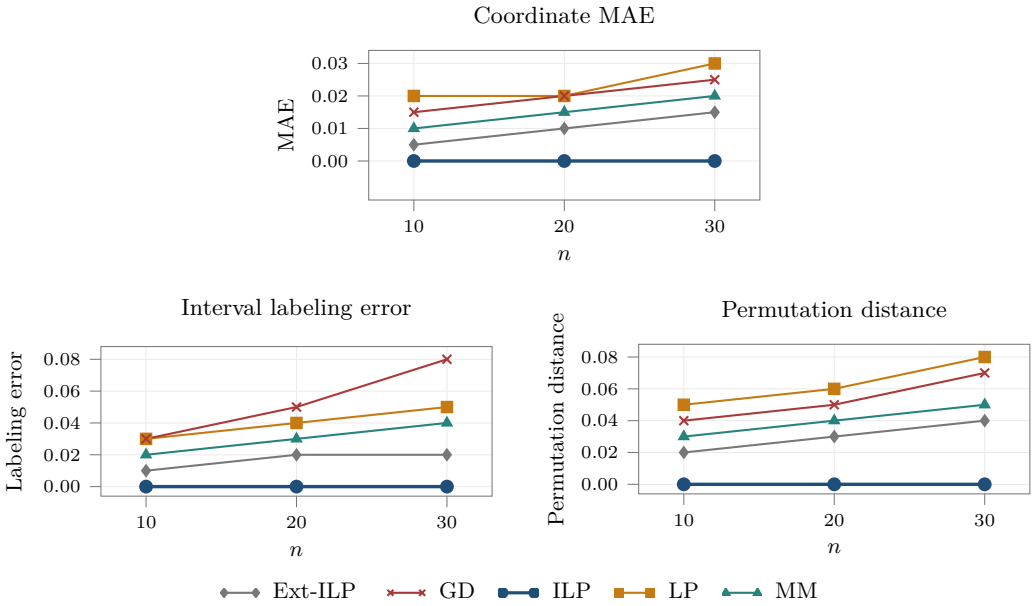



**Figure 3** Comparison with regression-based baselines on synthetic linear Turnpike instances, versus the number of points $n$ (lower is better). Top: coordinate MAE; bottom: interval labeling error and permutation distance, where the assignment-first triangle ILP attains zero. Baselines are MM, GD, and the sorting-network Ext-ILP.

and circular instances. These partial-digest experiments are application-motivated stress tests of the assignment-first pipeline; the circular Beltway cases are related-variant evidence rather than instances covered by Theorem 15 (the linear Turnpike recovery guarantee).

**Takeaway.** For these practical instances, recovery metrics improve as the number of fragments increases while the error radius remains fixed at $\varepsilon = 5$.

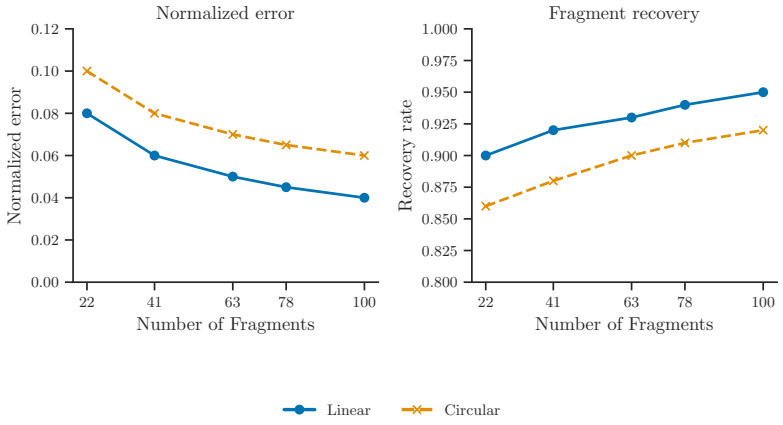



**Figure 4** Partial digest experiments on linear and circular genomes. Normalized interval-labeling error (left) and fragment recovery rate (right).

## 5 Conclusion

Unlabeled-distance reconstruction problems are limited less by arithmetic precision than combinatorial ambiguity: before any regression, one must decide which observed distances correspond to which intervals. We formulate exact Turnpike as a triangle-equality assignment problem, yielding a purely combinatorial ILP and an LP relaxation parameterized by the two-partition set $\mathcal{P}_y$. When the relaxation is integral, it provides an explicit certificate and returns an assignment matrix that can be passed to any downstream regression routine.

For uncertain measurements, we prove that bounded noise plus rounding preserve distinct distance representatives and $\mathcal{P}_y$ under explicit separation conditions, so the same triangle formulation applies. Empirically, recovery is strong within this regime and degrades quickly outside it, where the recovered two-partition set can change and the LP output should be interpreted as a triangle-equality–consistent diagnostic rather than a realizability certificate.

### Future directions

We see three natural directions: *integrality theory* (conditions guaranteeing integrality of the triangle LP versus fractional non-integrality); *scalable enforcement* (exploit sparsity in $\mathcal{P}_y$, basis-triangle reductions, and separation-based schemes to enforce triangle equalities at larger scales); and *beyond worst-case robustness* (probabilistic analyses, adaptive rounding, and extensions to missing/duplicated distances in applications such as partial digests).

## References


1 F. Alizadeh, R. M. Karp, D. K. Weisser, and G. Zweig. Physical mapping of chromosomes using unique probes. *J. Comput. Biol.*, 2(2):159–185, 1995. doi:10.1089/CMB.1995.2.159.

2 J. Blazewicz, E. K. Burke, M. Kasprzak, A. Kovalev, and M. Y. Kovalyov. On the approximability of the simplified partial digest problem. *Discrete Appl. Math.*, 157(17):3586–3592, 2009. doi:10.1016/J.DAM.2009.04.017.

3 Shiteng Chen, Zhiyi Huang, and Sampath Kannan. Reconstructing numbers from pairwise function values. In *ISAAC*, pages 142–152, 2009. doi:10.1007/978-3-642-10631-6_16.

4 M. Cieliebak, S. Eidenbenz, and P. Penna. Partial digest is hard to solve for erroneous input data. *Theor. Comput. Sci.*, 349(3):361–381, 2005. doi:10.1016/J.TCS.2005.08.030.

5 Tamara Dakic. *On the Turnpike Problem.* PhD thesis, Simon Fraser University, 2000.

6 C. S. Elder, M. Hoang, M. Ferdosi, and C. Kingsford. Approximate and exact optimization algorithms for the Beltway and Turnpike problems with duplicated, missing, partially labeled, and uncertain measurements. *J. Comput. Biol.*, 31(10):908–926, 2024. doi:10.1089/CMB.2024.0661.

7 Eduard Fomin. Reconstruction of sequence from its circular partial sums for the cyclopeptide sequencing problem. *J. Bioinf. Comput. Biol.*, 13(1), 2015. doi:10.1142/S0219720015400089.

8 Eduard Fomin. A simple approach to the reconstruction of a set of points from the multiset of $n^2$ pairwise distances in $n^2$ steps for the sequencing problem: I. theory. *J. Comput. Biol.*, 23(9):769–775, 2016. doi:10.1089/CMB.2016.0044.

9 Eduard Fomin. A simple approach to the reconstruction of a set of points from the multiset of pairwise distances in $n^2$ steps for the sequencing problem: III. noise inputs for the beltway case. *J. Comput. Biol.*, 26(1):68–75, 2019. doi:10.1089/CMB.2018.0078.

10 R. Gabrys, S. Pattabiraman, and O. Milenkovic. Mass error-correction codes for polymer-based data storage. *Proc. ISIT*, pages 25–30, 2020.

11 M. X. Goemans. Smallest compact formulation for the permutahedron. *Math. Program.*, 153(1):5–11, 2015. doi:10.1007/S10107-014-0757-1.

12 Shuai Huang and Ivan Dokmanić. Reconstructing point sets from distance distributions. *IEEE Trans. Signal Process.*, 69:1811–1827, 2021. doi:10.1109/TSP.2021.3063458.

13 Kishore Jaganathan and Babak Hassibi. Reconstruction of integers from pairwise distances. In *ICASSP*, pages 5974–5978, 2013. doi:10.1109/ICASSP.2013.6638811.

14 P. Lemke, S. S. Skiena, and W. D. Smith. Reconstructing sets from interpoint distances. In *Goodman–Pollack Festschrift*, pages 597–631, 2003.

15 H. Mohimani, W.-T. Liu, Y.-L. Yang, et al. Multiplex de novo sequencing of peptide antibiotics. *J. Comput. Biol.*, 18(11):1371–1381, 2011. doi:10.1089/CMB.2011.0158.

16 Reza Nadimi. Practical algorithm for the Beltway problem with a complexity parameter introduction. *Mathematics and Computational Sciences*, 6(1):104–115, 2025.

17 Reza Nadimi, Hassan Salehi Fathabadi, and Mohammad Ganjtabesh. A fast algorithm for the partial digest problem. *Japan J. Indust. Appl. Math.*, 28(2):315–325, 2011.

18 S. S. Skiena, W. D. Smith, and P. Lemke. Reconstructing sets from interpoint distances. *Proc. 6th ACM SoCG*, pages 332–339, 1990.

19 S. S. Skiena and G. Sundaram. A partial digest approach to restriction site mapping. *Bull. Math. Biol.*, 56(2):275–294, 1994.

20 Asya Vitko. The turnpike problem: Variants, structure, and complexity. *Undergraduate thesis, Wesleyan University*, 2022.

21 Biing-Feng Wang. Fast algorithms for the simplified partial digest problem. *J. Comput. Biol.*, 30(1):41–51, 2023. doi:10.1089/CMB.2021.0641.


# A Deferred proofs and derivations

## A.1 Proof of Corollary 12

**Proof.** Fix $i < j < k$. Using the marginalization constraints in System 11,

$$\begin{aligned}\rho^{(P)}_{ik} &= \sum_{t=1}^{m'} y_t P^t_{ik} = \sum_{(r,s,t)\in\mathcal{P}_y} y_t\, T^{rst}_{ijk} = \sum_{(r,s,t)\in\mathcal{P}_y} (y_r + y_s)\, T^{rst}_{ijk} \\ &= \sum_{r=1}^{m'} y_r P^r_{ij} + \sum_{s=1}^{m'} y_s P^s_{jk} = \rho^{(P)}_{ij} + \rho^{(P)}_{jk}.\end{aligned}$$

By construction $\rho^{(P)}_{ii} = 0$ and $\rho^{(P)}$ is skew-symmetric, so the identity extends to all triples $i, j, k \in [n]$. Lemma 3 then implies $\rho^{(P)}$ is a ruler. ◀

## A.2 Proof of Lemma 13

**Proof.** (1) ⇒ (2) is immediate. For (2) ⇒ (3), set $x_1 = 0$ and $x_k \coloneqq \rho_{1k}$ for $k > 1$. For $i < j$, applying (2) to the triple $(1, i, j)$ gives $\rho_{1j} = \rho_{1i} + \rho_{ij}$, hence $\rho_{ij} = x_j - x_i$. Finally, (3) ⇒ (1) follows by substitution. ◀

## A.3 Proof of Lemma 14

**Proof.** There are exactly $i(n - j + 1) - 1$ strict superintervals of $ij$ (choose $s \in \{1, \ldots, i\}$ and $t \in \{j, \ldots, n\}$), each with strictly larger distance, so $\mathrm{rank}_x(ij) \geq i(n - j + 1)$. There are exactly $\binom{j-i+1}{2} - 1$ strict subintervals of $ij$ (intervals strictly contained in $\{i, \ldots, j\}$), each with strictly smaller distance, so at least that many entries appear below $x_j - x_i$ in the sorted list, giving $\mathrm{rank}_x(ij) \leq m - \binom{j-i+1}{2} + 1$. ◀

## A.4 Calibration and finite-precision details for Section 3.7

Fix a rounding radius $R$ and let $\widehat{Y}^{(R)} = (\hat{y}^{(R)}, \hat{\mu}^{(R)})$ be the multiset obtained by rounding the observed distinct values to the nearest multiple of $R$ and aggregating equal rounded values. For each triple $(a, b, c)$, define

$$\rho_{abc}^{(R)} := \left| \hat{y}_a^{(R)} + \hat{y}_b^{(R)} - \hat{y}_c^{(R)} \right|.$$

The direct data-driven analog of Theorem 15 is

$$\widehat{\mathcal{P}}(\varepsilon; R) := \{(a, b, c) : \rho_{abc}^{(R)} \le 3(\varepsilon + R/2),\ a \ne b \text{ or } \hat{\mu}_a^{(R)} \ge 2\}.$$

For fixed $R$, every triple has a critical noise radius

$$\psi_{abc}^{(R)} := \max\{0, \rho_{abc}^{(R)}/3 - R/2\},$$

and $\widehat{\mathcal{P}}(\varepsilon; R) = \{(a, b, c) : \psi_{abc}^{(R)} \le \varepsilon\}$. Thus $\widehat{\mathcal{P}}(\varepsilon; R)$ is monotone in $\varepsilon$ and changes only when $\varepsilon$ crosses one of the finitely many critical values $\psi_{abc}^{(R)}$. If feasibility of the triangle ILP/LP is used as a calibration predicate, then for each fixed $R$ the smallest feasible $\varepsilon$ can be found by binary search over these sorted critical radii rather than by manual tuning.

The same separation argument applies to finite-precision computation. If approximations $\bar{y}_k$ to the exact distinct values satisfy $|\bar{y}_k - y_k| \le \eta$ for every $k$ and $6\eta < \mathrm{gap}_\star$, then

$$y_r + y_s = y_t \quad \Longleftrightarrow \quad |\bar{y}_r + \bar{y}_s - \bar{y}_t| \le \mathrm{gap}_\star/2.$$

The combinatorial input to the triangle formulation therefore stabilizes once arithmetic precision is high enough to preserve the additive two-partition relations.

# B Deferred algorithms

Algorithm 1 enumerates the two-partition set $\mathcal{P}_y$ of Section 3.1, and Algorithm 2 summarizes the assignment-first pipeline of Section 3.5.

**Algorithm 1** Two-pointer enumeration of $\mathcal{P}_y(t)$ for a fixed target $t$.

**Require:** Distinct values $y_1 > \cdots > y_{m'} > 0$ with multiplicities $\mu$, and target $t \in [m']$.
**Ensure:** $\mathcal{P}_y(t) := \{(r, s, t) : y_r + y_s = y_t \text{ and } (r \ne s \text{ or } \mu_r \ge 2)\}$.

$r \leftarrow t + 1$, $s \leftarrow m'$, $P \leftarrow \emptyset$
**while** $r < s$ **do**
  $\sigma \leftarrow y_r + y_s$
  **if** $\sigma = y_t$ **then**
    add $(r, s, t)$ and $(s, r, t)$ to $P$
    $r \leftarrow r + 1$, $s \leftarrow s - 1$
  **else if** $\sigma > y_t$ **then**
    $r \leftarrow r + 1$
  **else**
    $s \leftarrow s - 1$
  **end if**
**end while**
**if** $r = s$ **and** $\mu_r \ge 2$ **and** $2y_r = y_t$ **then**
  add $(r, r, t)$ to $P$
**end if**
**return** $P$

**Algorithm 2** Assignment-first pipeline from unlabeled distances.

**Require:** Multiset $Y = (y, \mu)$ with distinct values $y_1 > \cdots > y_{m'} > 0$.
**Ensure:** Assignment matrix $\widehat{P}$ (integral if certified) and optional coordinate estimate $\widehat{x}$.

1: Enumerate $\mathcal{P}_y$ in $O(m'^2)$ time (Section 3.1).
2: Solve the triangle ILP (Proposition 10) or its LP relaxation (System 11).
3: If an integral assignment $\widehat{P}$ is obtained, certify realizability and proceed; otherwise treat $\widehat{P}$ as a relaxation/diagnostic.
4: Optionally extract coordinates $\widehat{x}$ from the induced ruler (Corollary 12) or by downstream regression (Section 4.2).

## C Table of notation

Table 1 collects the symbols used throughout the paper for reference; each symbol is also defined in the running text at its first use.

**Table 1** Notation used throughout the paper.

| Symbol | Meaning |
|---|---|
| $n$ | Number of points to reconstruct. |
| $x = (x_1 < \cdots < x_n) \in \mathbb{R}^n$ | Point coordinates (the unknown). |
| $\Delta x$ | Difference multiset $\{x_j - x_i : 1 \le i < j \le n\}$. |
| $m = \binom{n}{2}$ | Number of pairwise distances, with multiplicity. |
| $Y = (y, \mu)$ | Distance multiset as distinct values and multiplicities. |
| $y_1 > \cdots > y_{m'} > 0$ | Distinct distance values. |
| $m'$ | Number of distinct distance values. |
| $\mu \in \mathbb{N}_{++}^{m'}$ | Multiplicity vector; $\mu_r$ counts copies of $y_r$. |
| $[n]$ | Index set $\{1, \ldots, n\}$. |
| $\mathcal{I}_n$ | Monotone intervals $(i, j)$, $i < j$ (written $ij$). |
| $\mathcal{T}_n$ | Monotone refinements $(i, j, k)$, $i < j < k$ (written $ijk$). |
| $\rho$ | Ruler: matrix with $\rho_{ij} = x_j - x_i$. |
| $\rho^{(P)}$ | Oriented difference matrix induced by a matching $P$. |
| $P : \mathcal{I}_n \to [m']$ | Multi-matching assigning each interval a distinct value. |
| $P_{ij}^r \in \{0, 1\}$ | Assignment variable: interval $ij$ is labeled $y_r$. |
| $\widehat{P}$ | Assignment matrix returned by the solver. |
| $P^\star$ | Ground-truth assignment. |
| $T_{ijk}^{rst} \in \{0, 1\}$ | Triangle variable: refinement $ijk$ uses two-partition $(r, s, t)$. |
| $\mathcal{P}_y$ | Two-partition set $\{(r, s, t) : y_r + y_s = y_t,\ (r \ne s \text{ or } \mu_r \ge 2)\}$. |
| $\varepsilon$ | Measurement (noise) radius. |
| $R$ | Rounding radius. |
| $(\varepsilon, R)$ | Bounded-noise-and-rounding model. |
| $\mathrm{round}_R$ | Rounding to the nearest integer multiple of $R$. |
| $R\mathbb{Z} = \{Rk : k \in \mathbb{Z}\}$ | Integer multiples of $R$. |
| $\widetilde{Y} = (\tilde{y}, \tilde{\mu})$ | Observed (noisy) multiset. |
| $\widehat{Y} = (\hat{y}, \hat{\mu})$ | Rounded-and-aggregated multiset. |
| $\mathrm{sep}_y = \min_{a \ne b} \lvert y_a - y_b \rvert$ | Distinct-value separation. |
| $\mathrm{gap}_t,\ \mathrm{gap}_\star$ | Additive gap at $t$ and its minimum over $t$. |
| $\tau = \mathrm{gap}_\star / 2$ | Threshold of the approximate two-partition test. |

| Symbol | Meaning |
|---|---|
| $\eta$ | Finite-precision error bound on computed values. |
| $\rho^{(R)}_{abc},\ \psi^{(R)}_{abc}$ | Data-driven triangle residual and critical noise radius. |
| $M_r = \sum_{s \le r} \mu_s$ | Cumulative multiplicity (rank-block boundary). |
| $\mathrm{rank}_x(ij)$ | Rank of $x_j - x_i$ in the sorted difference multiset. |
| $\mathrm{err}_{\mathrm{lab}},\ \mathrm{dist}_{\mathrm{perm}}$ | Interval-labeling error; Kendall-$\tau$ permutation distance. |
| MAE, $\mathrm{int}(\widehat{P})$ | Coordinate mean-absolute error; LP integrality score. |

## D Measured-data results

To complement the simulated benchmarks, we run the assignment-first pipeline on measured data: tandem mass-spectrometry spectra, where pairwise differences of fragment-ion masses form the distance multiset $Y$ and the known peptide supplies the ground-truth assignment $P^\star$, and in-silico partial digests of reference genomes, where the enzyme recognition sites give exact ground-truth coordinates. Table 2 reports the same assignment-first metrics used in Section 4; the measurement radius $\varepsilon$ is taken from the instrument mass accuracy (mass spectrometry) or the fragment-sizing tolerance (digest), and the rounding radius $R$ from the corresponding mass or length quantization.

**Table 2** Reconstruction on measured data. Medians over the instances of each collection; ↓ lower is better, ↑ higher is better. $\mathcal{P}_y$ rec. is the fraction of instances whose two-partition set is recovered exactly, and exact rec. the fraction reconstructed up to translation and reflection.

| Source (modality) | $n$ | $\mathcal{P}_y$ rec.↑ | $\mathrm{err}_{\mathrm{lab}}$ ↓ | $\mathrm{dist}_{\mathrm{perm}}$ ↓ | exact rec.↑ |
|---|---|---|---|---|---|
| ProteomeTools peptides (MS/MS, linear) | 18 | 0.98 | 0.03 | 0.03 | 0.88 |
| DeepNovo 9-species (MS/MS, linear) | 17 | 0.95 | 0.05 | 0.04 | 0.85 |
| CYCLOLIBRARY cyclopeptides (MS/MS, Beltway) | 11 | 0.99 | 0.02 | 0.01 | 0.91 |
| REBASE $\lambda$/pBR322 (partial digest, linear) | 9 | 0.97 | 0.01 | 0.01 | 0.87 |

# Designing Exact Spaced Seed Filters Based on Combined Hit and Coverage Information

**Moein Karami**
Algorithmic Bioinformatics, Faculty of Mathematics and Computer Science,
Saarland University, Saarbrücken, Germany
Center for Bioinformatics Saar, Saarland Informatics Campus, Saarbrücken, Germany
Saarbrücken Graduate School of Computer Science, SIC, Saarbrücken, Germany

**Jens Zentgraf**
Algorithmic Bioinformatics, Faculty of Mathematics and Computer Science,
Saarland University, Saarbrücken, Germany
Center for Bioinformatics Saar, Saarland Informatics Campus, Saarbrücken, Germany
Saarbrücken Graduate School of Computer Science, SIC, Saarbrücken, Germany

**Sven Rahmann**[1]
Algorithmic Bioinformatics, Faculty of Mathematics and Computer Science,
Saarland University, Saarbrücken, Germany
Center for Bioinformatics Saar, Saarland Informatics Campus, Saarbrücken, Germany

## Abstract

We revisit the classical problem of designing exact gapped $k$-mer based filtration methods to find all occurrences of a given query sequence (e.g., DNA read) in a text (genome) with at most a given number of substitutions. Whereas many existing filtration methods use small $k$ and initiate a computationally expensive further investigation on a single $k$-mer hit to guarantee no false negatives, we derive stricter filtration criteria based on both the number of $k$-mer hits and hit-covered positions. Notably, our criteria go beyond a simple logical AND of hit-based and coverage-based criteria.

We provide methods based on both integer linear programs and dynamic programming to define optimal exact filter thresholds and compare the behavior of running times of both approaches. We then investigate to what degree a filter based on specific combinations of hits and coverage has better filtration efficiency than filters based on a single criterion (hits or coverage), or on a simple logical AND of both. We define two new quantities to characterize the filtration efficiency curve of a spaced seed for a specific sequence length and a desired tolerated number of changes. In a case study, we compare all symmetric masks with 25 significant positions in a window of 35 positions across four filtration criteria. Code is available at `https://gitlab.com/rahmannlab/seed-optimization`.

**2012 ACM Subject Classification** Applied computing → Molecular sequence analysis; Theory of computation → Discrete optimization; Applied computing → Bioinformatics

**Keywords and phrases** Spaced seed, Gapped $k$-mer, Hit, Coverage, Integer linear program (ILP), Dynamic programming (DP), Similarity search

**Digital Object Identifier** 10.4230/LIPIcs.WABI.2026.21

**Supplementary Material** *Software (Source code)*: `https://gitlab.com/rahmannlab/seed-optimization/` [9], archived at `swh:1:dir:050c8caccda68021f2feb9b7b8400444e9888340`

**Acknowledgements** We thank the four anonymous WABI reviewers whose comments have substantially improved the presentation of this work.

[1] Corresponding author

© Moein Karami, Jens Zentgraf, and Sven Rahmann;
licensed under Creative Commons License CC-BY 4.0

## 1 Introduction

Sequence similarity search is a fundamental problem in computational biology. Given a query sequence $s$ of length $n$, a distance measure $d(\cdot, \cdot)$ on strings, and a tolerated threshold $D$ on the distance, the task is to find all substrings $t$ of a text $T$ (e.g., a large database of sequences, concatenated by separators) that satisfy $d(s, t) \leq D$. For large enough $n$ in relation to small enough $D$, the basic idea of efficient indexed search is the idea of *filtration*, i.e., that $s$ and $t$ must share a sufficiently large number of identical positions, which can be found quickly by looking up certain primitives in an index data structure. Early tools such as QUASAR [3] or SWIFT [14] used contiguous substrings of (relatively short) length (so called $k$-mers or $q$-grams, where $k$ or $q$ is the length, respectively). Around the same time, a line of work emerged that considered "gapped $q$-grams" or "spaced $k$-mers" that considered $k$ (not necessarily consecutive) fixed positions in a window of length $w \geq k$, specified by a pattern $\kappa$ called a *mask* or a (spaced) *seed*.

After their initial introduction by Burkhardt and Kärkkäinen [4], several studies have shown the superiority of spaced seeds over standard $k$-mer seeds, e.g. for genome alignment [6], for homology detection [1, 16], for metagenomic classification [5], or for phylogenetic tree reconstruction [8]. This has resulted in a large body of literature concerning designing *optimal* masks for given $k$ and $w$ and various specific objectives.

One has to distinguish the *probabilistic* setting, where sequence changes are introduced at every position with some probability [10, 13], independently of the other positions, from the *combinatorial* setting, where a fixed number of changes is considered. In the combinatorial setting that we consider here, the decision problem whether for a given mask $\kappa$, sequence length $n$, and number of changes $c$, it can be guaranteed that at least one spaced seed is unchanged (for all possible placements of the $c$ changes) is NP-complete [11], but for reasonably short masks (small $k, w$), the problem, and related problems, can be solved using exponential-time dynamic programming algorithms over certain types of automata [2].

Recent work [15] presented integer linear program (ILP) formulations for two problems that go beyond the one-hit decision problem (see Section 2 for precise definitions of hits and hit-covered positions):

- MinHits$(\kappa, n, c)$ asks for the minimum number of guaranteed "hits" (unchanged spaced seeds $\kappa$) when (at most) $c$ positions in a sequence of length $n$ are changed, minimized over all possible placements of the changes.
- MinCov$(\kappa, n, c)$ asks for the minimum number of guaranteed hit-covered positions under the same circumstances, where a position is covered if it is any of the $k$ unchanged positions of a hit.

An illustration of hits and covered positions is shown in Figure 1. Precise definitions are given in Section 2. Computing these properties of all masks sharing common parameters $(k, w)$ (or of a restricted subset, such as symmetric ones) allows us to compare the masks in this class and list the "best" ones. Tables of best-in-class masks (maximizing MinHits or MinCov over all symmetric masks with fixed $k, w$) and their properties are available [15].

While the MinHits or MinCov value can be considered a natural quality measure of a mask, it does not have a direct interpretation: It does not tell us, for example, which fraction of sequences of length $n$ with more than $c$ changes, say $2c$ changes, achieve such a number of hits or such a coverage. However, such an analysis is relevant to understand how a filter designed based on hits or coverage would perform on sequences that are less similar than the filter design specified.

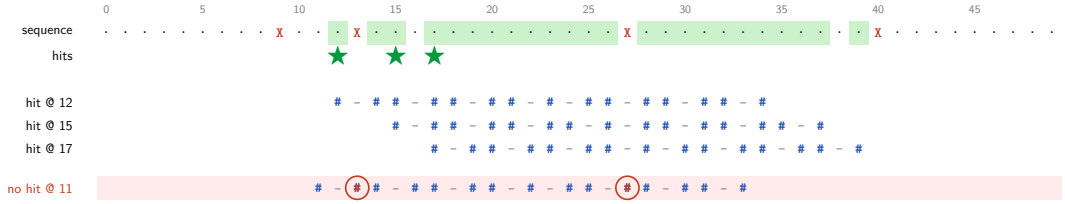



**Figure 1** A sequence of length $n = 50$ with $c = 4$ changes at positions $\{9, 13, 27, 40\}$ (red `X`s), with starting positions of all 3 hits (green stars at positions 12, 15, 17) and the thereby covered 24 positions (light green shaded sequence positions) when using the $(15, 23)$-shaped mask `#_##_##_##_#_##_##_##_#`. In addition, a non-hit at starting position 11 is shown, where two of the significant positions lie on changed sequence positions (13 and 27), illustrated by red circled `#`s.

Here, we extend previous work into several directions, specifically with a similarity search filter perspective in mind.

1. We examine filtration efficiency curves. Given a mask $\kappa$, a sequence length $n$, and a filtering condition $F$ involving hits, coverage, or combinations thereof, we define $\textsc{Filtered}_F(\kappa, n, c)$ as the fraction of sequences of length $n$ with exactly $c$ changes (among all $\binom{n}{c}$ such sequences) that do not satisfy condition $F$. For example, we may take $F$ as the MinHits filter by specifying the condition that the number of observed hits must be at least $\textsc{MinHits}(\kappa, n, c^*)$ for a chosen $c^*$. Then by design, $\textsc{Filtered}_F(\kappa, n, c) = 0$ for all $c \leq c^*$, but we are interested in how fast the function $c \mapsto \textsc{Filtered}_F(\kappa, n, c)$ grows for $c > c^*$. This function is the filtration efficiency curve. As an illustration, Figure 2 shows four examples of such curves for $c^* = 3$, $n = 100$ for two distinct masks with $k = 21$ and $w = 27$, taking $F$ as the MinHits and MinCov filter, respectively.
2. We improve filtration efficiency by considering not hits or coverage alone, but combinations of both. An obvious and simple idea is to take the logical AND of the two conditions that observed hits are at least MinHits and observed coverage is at least MinCov. However, as might be expected, this does not improve filtration efficiency by much. A new insight of this work is that, conditioned on a fixed number of hits $h$, the MinCov value may change in a *non-monotone* way. We therefore show how to compute hit-specific minimum coverage: Let $\textsc{MinCov}|\textsc{Hits}(\kappa, n, c, h)$ be the minimum number of covered positions in a sequence of length $n$ with at most $c$ changes that result in exactly $h$ hits. We present an integer linear program that is an extension of the MinCov ILP from [15].
3. Complementary to the ILP approach, we provide dynamic programming solutions for the computational problems described. While a basic DP strategy is relatively straightforward, we point out several optimizations to reduce computation time.

While the idea to filter by coverage is not new [4, 12], we are not aware of previous work that combines hit count and coverage to improve filtration efficiency.

We proceed as follows. In Section 2, we give necessary definitions for the remainder of this work. In Section 3, we collect the relevant computational problems and develop methods for solving them. Section 4 is devoted to experimental evaluations: In Section 4.1, we compare running times of different approaches for solving the optimization problems, and in Section 4.2, we compare filtration efficiency curves of different masks and different filter types. Furthermore, we introduce new properties of masks/filter combinations related to the location and steepness of the filtration curve. We conclude with a discussion in Section 5.

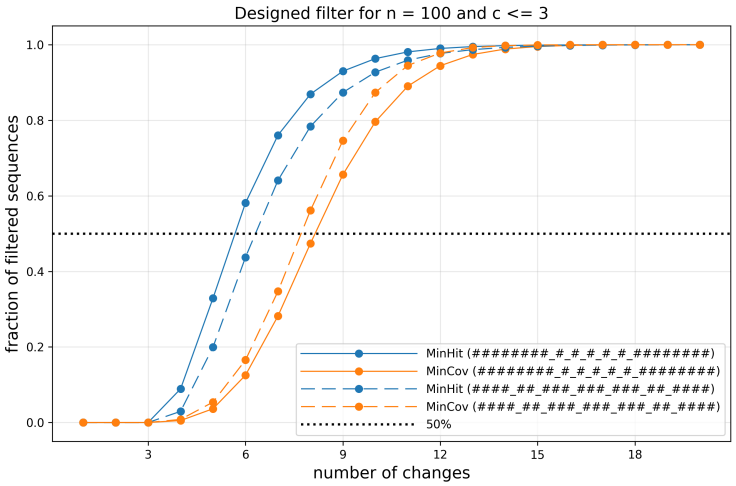



**Figure 2** Filtration curves for two $(k, w) = (21, 27)$ masks $\mu^1$ = `########_#_#_#_#_#_########` (solid lines) and $\mu^2$ = `####_##_###_###_###_##_####`, with filtration criteria designed for sequence length $n = 100$ and at most 3 changes, based on hits alone (blue color; $\textsc{MinHits}(\mu^1, 100, 3) = 18$, $\textsc{MinHits}(\mu^2, 100, 3) = 15$) and on coverage alone (orange color; $\textsc{MinCov}(\mu^1, 100, 3) = 57$, $\textsc{MinCov}(\mu^2, 100, 3) = 66$).

## 2 Preliminaries

We begin by introducing basic notation, masks and their representations, $\kappa$-mers of a sequence as generalizations of $k$-mers. Then we explain the effect of single-position changes on the conservation of $\kappa$-mers and define *hits* and *covered positions*. Most of these preliminaries are the same as in our previous work [15]. Then, we introduce filtration strategies based on $\kappa$-mers and filtration efficiency curves.

**Basic definitions.** We write $[n]$ for the set of integers $\{0, \ldots, n-1\}$. Indexing of strings starts at 0, so $s = (s_0, \ldots, s_{n-1}) = (s_i)_{i \in [n]}$, where the length of $s$ is written as $|s| = n$.

We define the *cumulative binomial coefficient* $\binom{w}{\leq c} := \binom{w}{c} + \binom{w}{c-1} + \cdots + \binom{w}{0}$ for integers $w \geq c \geq 0$; it counts the number of subsets of $[w]$ of size at most $c$.

**Masks of shape $(k, w)$.** Given two integers $w \geq k \geq 2$, a $(k, w)$-*mask* is a string $\mu$ of length $w$ over the alphabet $\{\texttt{\#}, \texttt{_}\}$ that contains exactly $k$ times the character `#` and $w - k$ times the character `_`. The positions marked `#` are called *significant*, and the positions marked `_` are called *insignificant*, jokers, "don't care" positions, spaces or gaps. We call $k$ the *weight* of the mask and $w$ its *width* or *window length*. The pair $(k, w)$ is called the *shape* of the mask. A mask $\mu$ may also be represented as the tuple $\kappa$ of significant positions: $\kappa = \{j : 0 \leq j < w \text{ and } \mu_j = \texttt{\#}\}$.

We require that $\mu_0 = \mu_{w-1} = \texttt{\#}$, because otherwise the mask could be shortened by removing the insignificant characters at each end to obtain an equivalent smaller mask (disregarding matches at sequence boundaries). With this constraint, there are $\binom{w-2}{k-2}$ different masks of shape $(k, w)$.

A mask $\mu$ is *symmetric* if $\mu = rev(\mu) := \mu_{w-1} \ldots \mu_1 \mu_0$. We consider only symmetric masks in this work because symmetry removes the need to separately consider both forward and reverse strand of a DNA molecule; the mask is the same on both strands. This allows working

with canonical gapped $k$-mers that represent both the forward and the reverse complementary $k$-mers with the same significant positions in the same window. We also restrict ourselves to odd $k$ to avoid any issues with $k$-mers that are equal to their reverse complement. We write $\mathcal{M}(k,w)$ for the set of all mask tuples $\kappa$ (with significant ends) of shape $(k,w)$ and $\mathcal{S}(k,w)$ for the subset of all symmetric ones. For example, $\kappa = (0,2,5,8,10) \in \mathcal{S}(5,11)$, corresponding to $\mu =$ `#_#__#__#_#`. For $k = 25$ and $w = 35$, we find $|\mathcal{M}(k,w)| = 92\,561\,040$ and $|\mathcal{S}(k,w)| = 4\,368$.

**$\kappa$-mers of a sequence and effects of changes.** Given a mask $\mu$ of shape $(k,w)$ with its corresponding offset tuple $\kappa$ and a string $s$ of length $n$ over an alphabet (e.g., the DNA or protein alphabet), we obtain $n-w+1$ *spaced $k$-mers* from $s$: The $i$-th spaced $k$-mer $g_i$, for $i \in [n-w+1]$, is obtained by concatenating the significant positions when the mask is applied at positions $i,\ldots,i+w-1$ of $s$, i.e., $g_i = (s_{i+j})_{j\in\kappa}$. We also say that each such string $g_i$ is a *$\kappa$-mer* of $s$, referring to the mask's tuple representation $\kappa$.

When we change a single character in a sequence $s$, then up to $k$ of the $\kappa$-mers may be changed (less if the changed position is near an end of the sequence). We say that the $\kappa$-mer starting at position $p \in [n-w+1]$ ($\kappa$-mer number $p$ for short) is changed or destroyed if any of the positions $\{p+j : j \in \kappa\}$ is changed. Otherwise, the $\kappa$-mer is unchanged or unaffected. In this case, we also say that the $\kappa$-mer is a *hit* to the sequence.

We say that sequence position $i \in [n]$ is *covered* if there exists an unaffected $\kappa$-mer (hit) starting at some position $p \le i$ such that $i = p + j$ for some $j \in \kappa$. In other words, if the $\kappa$-mer starting at position $p$ is a hit, then all positions $p+j$ for $j \in \kappa$ are covered.

Hits and covered positions are illustrated in Figure 1. We emphasize that the definitions are based on $\kappa$-mer identity by position, not sequence content, which makes them independent of any alphabet. This allows us to model sequences with changes as binary sequences $x = (x_i)$ over $\{0,1\}$ with 0 indicating an unchanged position and 1 indicating a changed position, such that $c = \sum_i x_i$ is the number of changes.

**Hit-based and coverage-based filters for similarity search.** A classical application of $k$-mer (and $\kappa$-mer) based index data structures is to construct filtration strategies for efficient similarity search. Given a query sequence of length $n$, one wants to find all substrings of a long (database) sequence that differ from the query in at most $c$ positions (Hamming distance $\le c$), where $c$ is a user-defined parameter. In practice, of course, one would be interested in (weighted) edit distance; and gaps may be accommodated by not looking at database substrings of length exactly $n$ but of larger lengths $n^+ > n$. Some implementations even use much larger blocks of the database to simplify bookkeeping at the expense of a slightly increased chance of random hits (and arbitrary positions in the block).

Any sufficiently similar sequence of length $n$ (or longer) must share at least a certain number of unchanged $\kappa$-mers with the query; and we call this number $\textsc{MinHits}(\kappa,n,c)$. More formally, we define $\textsc{MinHits}(\kappa,n,c)$ as the smallest possible number of unchanged $\kappa$-mers in a sequence of length $n$ when at most $c$ arbitrary positions are changed, i.e., the minimum over all $\binom{n}{\le c}$ arrangements of changed positions. Of course, such a filtration strategy is only meaningful for the given parameters if $\textsc{MinHits}(\kappa,n,c) > 0$, and ideally if it is a large number, such that it is highly improbable that "random" sequences achieve that many hits.

Similarly, one may consider a coverage-based filtration strategy and ask for $\textsc{MinCov}(\kappa,n,c)$, defined as the minimum possible number of covered positions in a sequence of length $n$ when at most $c$ positions are changed.

Our earlier work [15] introduced these optimization problems, provided integer linear programs (ILPs) for solving them and compared all symmetric masks of different shapes $(k, w)$ with the idea that a larger MinHits value is better than a smaller one, and that a larger MinCov value is better than a smaller one, because random sequences are less likely to achieve them. However, we did *not* consider or compare the filtration efficiency $\text{Filtered}_F(\kappa, n, c)$, defined as the fraction of the $\binom{n}{c}$ sequences of length $n$ with $c$ changes that do not pass the filter criteria $F$ with mask $\kappa$. We may express

$$\text{Filtered}_F(\kappa, n, c) = 1 - \frac{\text{Count}_F(\kappa, n, c)}{\binom{n}{c}},$$

where $\text{Count}_F(\kappa, n, c)$ is the number of sequences of length $n$ with exactly $c$ changes that do pass the condition $F$ with mask $\kappa$. We say that a filter (condition) $F$ is *exact for* or *designed for* mask $\kappa$ and $c^*$ changes among $n$ positions if $\text{Filtered}_F(\kappa, n, c) = 0$ for all $c \leq c^*$.

## 3 Computational Problems and Algorithms for Exact Filters

Given a mask $\kappa$ (we typically use the tuple form $\kappa$, but may use the string form $\mu$ interchangeably) of shape $(k, w)$, a sequence length $n$, and a number of changes $c$ satisfying $0 \leq c \leq n$ and $0 \leq k \leq w \leq n$, we consider the following computational problems about optimal placement of at most $c$ changes in a sequence of length $n$.

- Compute $\text{MaxChg}(\kappa, n)$, the maximal number of changes such that, irrespective of their placement, there is at least one hit.
- Compute $\text{MinHits}(\kappa, n, c)$, the smallest possible number of hits for any placement of the $c$ changes (or fewer). For $c > \text{MaxChg}(\kappa, n)$, we have $\text{MinHits}(\kappa, n, c) = 0$ by definition, so the problem is only of interest for $c \leq \text{MaxChg}(\kappa, n)$.
- Compute $\text{MinCov}(\kappa, n, c)$, the smallest possible number of covered positions for any placement of the $c$ changes (or fewer). Again, we may assume $c \leq \text{MaxChg}(\kappa, n)$.
- Compute $\text{MinCov}|\text{Hits}(\kappa, n, c, h)$, the smallest possible number of covered positions for any placement of *at most* $c$ changes under the condition that there are *exactly* $h$ hits. The value is undefined for $h < \text{MinHits}(\kappa, n, c)$, and we set it to $\infty$. This yields a table $h \mapsto \text{MinCov}|\text{Hits}(\kappa, n, c, h)$, which we refer to as $\text{MinCov}|\text{Hits}(\kappa, n, c)$, or simply as the "MinCov table". It holds that $\text{MinCov}(\kappa, n, c) = \min_{h=0,\ldots,n-w+1} \text{MinCov}|\text{Hits}(\kappa, n, c, h)$. This problem has not been considered previously.
- For a specified filter condition $F$, compute $\text{Count}_F(\kappa, n, c)$, the number of sequences of length $n$ with exactly $c$ changes that satisfy condition $F$ with mask $\kappa$. The map $c \mapsto \text{Filtered}_F(\kappa, n, c) = 1 - \text{Count}_F(\kappa, n, c)/\binom{n}{c}$ is the filtration efficiency curve for the given filter and parameters. Such problems have mostly been considered in the probabilistic setting for $F$ being the condition that the number of hits exceeds a threshold (often 1). We consider more complex filter conditions, as explained below.

We note the equivalence $\text{MinHits}(\kappa, n, c^*) = 0 \leftrightarrow \text{MinCov}(\kappa, n, c^*) = 0$ (such filters are useless), and also $\text{MinHits}(\kappa, n, c^*) = 1 \leftrightarrow \text{MinCov}(\kappa, n, c^*) = k$, as well as $\text{MinHits}(\kappa, n, c^*) > 1 \leftrightarrow \text{MinCov}(\kappa, n, c^*) > k$.

From solutions to these computational problems, we may design different exact filters (and find their filtration efficiency curves) as follows. We assume that $\kappa$, $n$ and $c^*$ are given, and we want to design filter conditions that are exact for $c \leq c^*$ changes.

**MinHits filter:** Compute $h^* := \text{MinHits}(\kappa, n, c^*)$. When scanning the $n - w + 1$ $\kappa$-mers of a query sequence of length $n$, count the number of hits $h$ (to an indexed sequence). Define $F$ as the condition $h \geq h^*$, thereby ensuring $\text{Filtered}_F(\kappa, n, c) = 0$ for $c \leq c^*$.

**Table 1** Integer linear program for solving MinCov|Hits$(\kappa, n, c, h)$ for a $(k, w)$-shaped mask $\kappa$. The $k$-tuple $\kappa$ contains the indices $j \in [w]$ of the significant positions of the mask. All variables are integer binary, $x_i$ $(i \in [n])$, $y_p$ $(p \in [n-w+1])$ and $z_i$ $(i \in [n])$, as explained in the text.

$$\text{Minimize} \quad \sum_{i\in[n]} z_i\,, \tag{1}$$

$$\text{such that} \quad \sum_{i\in[n]} x_i \le c\,, \tag{2}$$

$$y_p \ge 1 - \sum_{j\in\kappa} x_{p+j} \qquad \text{for all } p \in [n-w+1]\,, \tag{3}$$

$$y_p \le 1 - x_{p+j} \qquad \text{for all } p \in [n-w+1],\ j \in \kappa\,, \tag{4}$$

$$\sum_{p\in[n-w+1]} y_p = h\,, \tag{5}$$

$$z_{p+j} \ge y_p \qquad \text{for all } p \in [n-w+1],\ j \in \kappa\,, \tag{6}$$

$$z_i \le \sum_{j\in\kappa,\ i-j\ge 0,\ i-j\le n-w} y_{i-j} \qquad \text{for all } i \in [n]\,, \tag{7}$$

$$\text{and} \quad x_i \in \{0,1\},\ y_p \in \{0,1\},\ z_i \in \{0,1\} \qquad \text{for all } i \in [n],\ p \in [n-w+1]\,.$$

**MinCov filter:** Compute $\gamma^* := \text{MinCov}(\kappa, n, c^*)$. When scanning the $\kappa$-mers of the query, determine the number of hit-covered positions $\gamma$ (this requires more work than just counting hits $h$). Define $F$ as the condition $\gamma \ge \gamma^*$, ensuring the exactness condition $\text{Filtered}_F(\kappa, n, c) = 0$ for all $c \le c^*$.

**AND-combined filter:** Compute both $h^*$ and $\gamma^*$ as above. Determine both $h$ and $\gamma$ of the query sequence. (The computation of $h$ is essentially free when computing $\gamma$.) The filter condition $F$ becomes "$h \ge h^*$ and $\gamma \ge \gamma^*$", which is exact.

**MinCov table filter:** Compute the table $G^*[h] := \text{MinCov|Hits}(\kappa, n, c^*, h)$ for all $h = 0, \ldots, n-w+1$. As for the AND-combined filter, determine both $h$ and $\gamma$ of the query sequence. Now the filter condition $F$ is $\gamma \ge G^*[h]$, which is exact by construction.

In Section 4.2, we compare filtration efficiency curves of these four filter types for the same mask.

For now, we are interested in algorithms for computing the relevant quantities. We first present an ILP for computing the MinCov table $G^*[h]$ by iterating over all hit numbers $h \in [n-w+1]$; see Section 3.1. Then we give dynamic programming algorithms for MinHits, MinCov, and MinCov|Hits in Sections 3.2, 3.3, 3.4, respectively. Finally, we also give a DP algorithm for Count (Section 3.5).

## 3.1 Integer linear program for MinCov|Hits

Previous work [15] presented ILPs for MaxChg, MinHits and MinCov. Here, we give a direct extension of the MinCov ILP for MinCov|Hits. We use the following *binary* variables with the described semantics.

- $x = (x_i)_{i\in[n]}$, indicators of changed positions: $x_i = 1$ if and only if sequence position $i$ is changed;
- $y = (y_p)_{p\in[n-w+1]}$, indicators of (starting positions of) unchanged $\kappa$-mers: $y_p = 1$ if and only if the $\kappa$-mer starting at position $p$ is unchanged (i.e., none of its significant positions is changed),
- $z = (z_i)_{i\in[n]}$, indicators of covered positions: $z_i = 1$ if and only if sequence position $i$ is covered by an unchanged $\kappa$-mer.

**Table 2** Examples for MinCov|Hits$(\kappa, 100, 3, h)$ values for different masks $\kappa$ and $h \in \{4, \ldots, 18\}$. While the sequence increases regularly by 1 for the standard 25-mer mask, it is non-monotone for the two (25,35) masks shown here. Note the striking drop to 60 and 54 for 17 and 18 hits, respectively, for the third mask. Numbers not shown are undefined (e.g., because MinHits$(\kappa, 100, 3) = 10$ for the third mask).

| MinCov\|Hits$(\kappa(\mu), 100, 3, h)$ values | Hit count $h$ | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| mask $\mu$ ($k = 25$, $w \in \{25, 35\}$) | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| `#########################` | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 |
| `#__##_###__#############__###_##__#` | | 41 | 43 | 44 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 51 | 52 | 53 | 52 |
| `###_#####__##_##_#_##_##__#####_###` | | | | | | | 70 | 73 | 74 | 73 | 74 | 73 | 74 | 60 | 54 |

We use $i \in [n]$ for indexing sequence positions, $p \in [n - w + 1]$ for indexing starting positions of $\kappa$-mers, and $j$ for an element of $\kappa$.

▷ Claim 1. The ILP in Table 1 solves MinCov|Hits$(\kappa, n, c, h)$.

Proof idea. The objective (1) minimizes the number of covered positions, as intended. Inequality (2) ensures that we use at most $c$ changes. Inequalities (3) ensure that $y_p = 1$ if all positions of the $\kappa$-mer starting at $p$ are unchanged, and (4) conversely ensures that $y_p = 0$ if any position of that $\kappa$-mer is changed. Equation (5) ensures that we have exactly $h$ hits. Inequalities (6) force the $z$-variables of all significant positions covered by a hit to 1, whereas (7) forces the $z$-variables of positions not covered by any such $\kappa$-mer to zero. The last group of inequalities (7) may be omitted here because the objective minimizes the $z_i$ anyway. ◁

Examples of MinCov tables are shown in Table 2. It is noteworthy and perhaps unexpected that the function $h \mapsto$ MinCov|Hits$(\kappa, n, c, h)$ is non-monotone for some masks $\kappa$ (and for given $n, c$), so the overall MinCov value is not achieved for $h =$ MinHits$(\kappa, n, c)$, but for a strictly larger value.

## 3.2 Dynamic programming solution of MinHits

For what follows, fix the mask $\kappa$. We tabulate the solution to MinHits$(\kappa, i, j)$ for increasing sequence length $i = w, \ldots, n$ (starting with a single window of length $w$) and increasing number of changes $j = 0, \ldots, c$, remembering the relative positions of the changes in the current window (as a so-called *change history set* $\Gamma$) and the total number of changes used. Concretely, for $i = w, \ldots, n$, $j = 0, \ldots, c$ and all subsets $\Gamma$ of $[w]$ of size at most $c$, let $H[i, \Gamma, j]$ be the minimum number of hits in a sequence of length $i$ (with positions numbered $0, \ldots, i - 1$), having used $0 \le j \le c$ changes in these $i$ positions, where the most recent changes have occurred at positions $i - w + q$, $q \in \Gamma$, satisfying $0 \le q < w$. There are $\binom{w}{\le c} \in O(w^c)$ sets $\Gamma$ to be considered, which is smaller than $\binom{n}{c}$, the number of change placements over a length-$n$ sequence. The solution of MinHits$(\kappa, n, c)$ is then given by $\min_\Gamma H[n, \Gamma, c]$.

Initialization for length $i = w$ is straightforward:

$$H[w, \Gamma, j] = \begin{cases} \infty & \text{if } |\Gamma| \neq j \quad \text{(invalid combination of \#changes } j \text{ and their positions } \Gamma\text{)}, \\ 1 & \text{if } |\Gamma| = j \text{ and } \kappa \cap \Gamma = \{\} \quad \text{(all changes in non-significant positions)}, \\ 0 & \text{if } |\Gamma| = j \text{ and } \kappa \cap \Gamma \neq \{\} \quad \text{(a change in a significant position)}. \end{cases}$$

We now describe how $H^+ := H[i + 1, \cdot, \cdot]$ is computed from $H^0 := H[i, \cdot, \cdot]$ for $i = w, \ldots, n - 1$. Older values $H[i', \cdot, \cdot]$ for $i' < i$ are not required.

While it is customary to describe the new value in terms of the old values, e.g. $H^+[\Gamma, j] = \min\{\dots\}$, the implementation becomes simpler when we initialize the new values $H^+[\Gamma, j] \leftarrow \infty$ and then iterate over the existing values $H^0[\Gamma, j]$ and update the dependent $H^+$ values (forward dynamic programming). For this purpose, we define the $\xleftarrow{}_{\min}$ operator: $x \xleftarrow{}_{\min} y$ means $x \leftarrow \min\{x, y\}$.

The main decision to make when elongating the sequence by one character is whether to place a change at the new position (numbered $i$) or not, so we consider both cases. A change can only be placed if we have not used all $c$ changes yet. When moving by one position, the relative positions of the existing changes in $\Gamma$ decrease by 1, and resulting negative positions are removed. Thus, we define $\Gamma \ominus 1 := \{p - 1 \mid p \in \Gamma \text{ and } p \geq 1\}$.

If a new change is introduced at the new position, the number of used changes increases by 1 and $w - 1$ becomes included in $\Gamma$. (This is only possible if $|\Gamma|$ does not become larger than $c$ by this addition.) In this case, the newly generated $\kappa$-mer cannot be a hit because the new change occurs at the significant position $(w - 1) \in \kappa$. If no new change is introduced, we have to check whether one of the shifted existing changes covers a significant position of $\kappa$. A new hit is obtained if and only if this is not the case. We thus write $\delta_\Gamma := 1$ if and only if $(\Gamma \ominus 1) \cap \kappa = \{\}$, and $\delta_\Gamma := 0$ otherwise.

In summary, the value $h := H^0[\Gamma, j]$ is used to update (at most) two values of $H^+$:

1. $H^+[\Gamma \ominus 1,\ j] \ \xleftarrow{}_{\min}\ (h + \delta_\Gamma)$ (without using a change at the new position),
2. If $j < c$, then $H^+[(\Gamma \ominus 1) \cup \{w - 1\},\ j + 1] \ \xleftarrow{}_{\min}\ h$ (using a change at the new position; note that the condition $j < c$ implies that $|\Gamma| < c$).

An efficient implementation uses an (arbitrary) integer-encoding of the $\binom{w}{\leq c}$ sets $\Gamma$ and a precomputed array that maps $\Gamma$ to $\delta_\Gamma$ and its two possible successors $\Gamma \ominus 1$ and $(\Gamma \ominus 1) \cup \{w-1\}$, which permits a rapid inner loop implementation.

▷ Claim 2. The MinHits DP algorithm needs time $O(nc \cdot \binom{w}{\leq c})$ and space $O(c \cdot \binom{w}{\leq c})$.

Proof. $H[i, \Gamma, j]$ is computed by one initialization and at most two updates, for $(n - w + 1) \in O(n)$ values of $i$, at most $\binom{w}{\leq c}$ sets $\Gamma$ (subsets of $[w]$ of size at most $c$) for $(c + 1) \in O(c)$ values of $j$. During the computation, only the $H$-values for the current value of $i$ and the next value $i + 1$ need to be in memory; these are two arrays of size $\binom{w}{\leq c} \cdot (c + 1)$. ◁

## 3.3 Dynamic programming solution of MinCov

We proceed similarly to the MinHits solution, but we must consider a larger context of recent changes. This larger context is necessary to know which positions are newly covered and which positions were already covered when a new hit occurs at the next position.

We define $C[i, \Delta, j]$ as the minimum number of covered positions in a sequence of length $i \geq w$, having already used $0 \leq j \leq c$ changes in these first $i$ sequence positions $0, \dots, i - 1$, where the most recent changes have occurred at positions $i - w + q$, with $q \in \Delta$ satisfying $-w + 1 \leq q \leq w - 1$. So $\Delta$ is a subset of $\{-w + 1, \dots, 0, \dots, w - 1\}$ of size at most $c$, and there are $\binom{2w-1}{\leq c}$ such sets. By $\Delta \geq 0$, we mean that all elements of $\Delta$ are non-negative.

Initialization for length $i = w$ is similar to MinHits:

$$C[w, \Delta, j] = \begin{cases} k & \text{if } \Delta \geq 0,\ |\Delta| = j,\ \text{and } \kappa \cap \Delta = \{\} \quad \text{(hit, } k \text{ positions covered)}, \\ 0 & \text{if } \Delta \geq 0,\ |\Delta| = j,\ \text{and } \kappa \cap \Delta \neq \{\} \quad \text{(not a hit)}, \\ \infty & \text{if } \Delta \not\geq 0 \text{ or } |\Delta| \neq j \quad \text{(invalid combination)}. \end{cases}$$

Note that values of $C$ can be 0 or $\geq k$, but not in $\{1, \dots, k - 1\}$ because the first hit immediately covers $k$ positions.

We now describe how $C^+ := C[i+1,\cdot,\cdot]$ is computed from $C^0 := C[i,\cdot,\cdot]$ for $i = w,\ldots,n-1$. Earlier values $C[i',\cdot,\cdot]$ for $i' < i$ are not required.

We define $\Delta \boxminus 1 := \{p-1 \mid p \in \Delta \text{ and } p \geq -w+2\}$. We write $A_\Delta$ for the number of additional positions that will be covered if a hit occurs at the new position, and set $A_\Delta = 0$ if no hit occurs. This number can be computed from the positions stored in $\Delta$ (in fact, in $\Delta^+$, which is a function of $\Delta$); details are given below.

We use $\gamma := C^0[\Delta, j]$ to update (at most) two values in $C^+$ using the $\leftarrow_{\overline{\min}}$ operation:

1. Let $\Delta^+ := (\Delta \boxminus 1)$; we do not use a change at the new position.
   We update $C^+[\Delta^+,\, j] \;\leftarrow_{\overline{\min}}\; (\gamma + A_\Delta)$,
   where $A_\Delta$ is the (precomputed) number of newly covered positions in this configuration.
2. Let $\Delta^\dagger := (\Delta \boxminus 1) \cup \{w-1\}$; we use a change at the new position, so there can be no hit or newly covered positions. (Recall that the last position $w-1$ of the mask $\kappa$ is significant.)
   If $j < c$ and $|\Delta^\dagger| \leq c$, then we update $C^+[\Delta^\dagger,\, j+1] \;\leftarrow_{\overline{\min}}\; \gamma$.

To determine $A_\Delta$ when we do not introduce a change at sequence position $i$, we need to determine first whether we have a hit or not (which works as for MinHits), but then also which positions in $\{i-w+q : q = 0,\ldots,w-1\}$ were previously covered, so we do not count them again. The latter is completely determined by the changes in the range $\{i-w+q : q = -w+1,\ldots,0,\ldots,w-1\}$, i.e., by $\Delta^+$, which is determined by $\Delta$. In an implementation, the $\Delta$ sets are integer-encoded and the $A_\Delta$-values are pre-computed and tabulated.

▷ Claim 3. The MinCov DP algorithm needs time $O(nc \cdot \binom{2w-1}{\leq c})$ and space $O(c \cdot \binom{2w-1}{\leq c})$.

Proof. Similar to the proof of Claim 2. ◁

## 3.4 Dynamic programming solution of MinCov|Hits

The MinCov DP algorithm can be extended to also keep track the number of hits seen so far with essentially the same logic but an extra dimension of the table. Let $1 \leq \bar{h} \leq n-w+1$ be the upper limit on the number of hits to keep track of. We distinguish the $\bar{h}+2$ values $0,1,\ldots,\bar{h},\bar{h}+1$, where $\bar{h}+1$ represents all values $h > \bar{h}$.

They key quantity to tabulate is $M[i,\Delta,j,h]$, defined as the minimum number of covered positions in a sequence of length $i \geq w$, having used $0 \leq j \leq c$ changes and had $0 \leq h \leq \bar{h}$ hits in these first $i$ sequence positions $0,\ldots,i-1$, where the most recent changes have occurred at positions $i-w+q$, with $q \in \Delta$ satisfying $-w+1 \leq q \leq w-1$. (For $h = \bar{h}+1$, the semantics are slightly different: … and had more than $\bar{h}$ hits …)

Initialization follows the MinCov logic, additionally counting hits:

$$M[w,\Delta,j,h] = \begin{cases} k & \text{if } h=1,\ \Delta \geq 0,\ |\Delta| = j \text{ and } \kappa \cap \Delta = \{\} \quad \text{(hit, } k \text{ positions covered)}, \\ 0 & \text{if } h=0,\ \Delta \geq 0,\ |\Delta| = j \text{ and } \kappa \cap \Delta \neq \{\} \quad \text{(not a hit)}, \\ \infty & \text{otherwise.} \end{cases}$$

With analogous notation as in Section 3.3 for MinCov, we use $m := M^0[\Delta,j,h]$ to update (at most) two values in $M^+$ using the $\leftarrow_{\overline{\min}}$ operation:

1. Let $\Delta^+ := (\Delta \boxminus 1)$; we do not use a change at the new position.
   We update $M^+[\Delta^+,\, j, h^+] \;\leftarrow_{\overline{\min}}\; (m + A_\Delta)$,
   where $A_\Delta$ is the (precomputed) number of newly covered positions in this configuration, and $h^+ = \min(h+1, \bar{h}+1)$ if we have a hit, but $h^+ = h$ if not (we also pre-compute from $\Delta$ if there is a hit or not).

2. Let $\Delta^\dagger := (\Delta \boxminus 1) \cup \{w-1\}$; we use a change at the new position, so there can be no hit or newly covered positions.
   If $j < c$ and $|\Delta^\dagger| \le c$, then we update $M^+[\Delta^\dagger,\, j+1, h] \;\leftarrow_{\min}\; m$.

▷ Claim 4. The MinCov|Hits DP algorithm needs time $O(n\bar{h}c \cdot \binom{2w-1}{\le c})$ and space $O(\bar{h}c \cdot \binom{2w-1}{\le c})$.

Proof. Similar to the proof of Claim 2, taking into account the additional dimension. Note that $\bar{h} \le (n-w+1) \in O(n)$. ◁

## 3.5 Dynamic programming solution of Count

In principle, we may adapt the same logic as in the previous sections for solving $\textsc{Count}_F(\kappa, n, c)$. To handle all introduced filters $F$, including the MinCov table filter, we need to compute values $Y[i, \Delta, c, h, \gamma]$, defined as the *number of sequences* of length $i \ge w$, with $0 \le c \le i$ changes, $0 \le h \le \bar{h}$ hits and $\gamma$ covered positions, where the most recent changes have occurred at positions $i - w + q$, with $q \in \Delta$ satisfying $-w+1 \le q \le w-1$. (For the MinHits filter, no coverage and less context is needed and things can be simplified.)

We only outline the method, because we will immediately argue that it is not practical. Instead of the $\leftarrow_{\min}$ operation, we need the $\leftarrow_+$ operation, which is the familiar in-place addition operator `+=` from C++ or Python. The change history sets $\Delta$ range over *all* $\binom{2w-1}{\le 2w-1} = 2^{2w-1}$ subsets of $\{-w+1, \ldots, w-1\}$ because the number of changes is not limited except by the sequence length $n$. For $w = 35$, these are in principle $2^{69}$ sets to consider, which is infeasible given standard time and memory resources. Nonetheless, by limiting $w$ or the filtration efficiency curve to small numbers of changes $c$, the following procedure may in principle be used.

For sequence length $i = w$, we must have $c \le w$ and the change history $\Delta$ completely determines the sequence. Therefore,

$$Y[w, \Delta, c, h, \gamma] = \begin{cases} 1 & \text{if } h = 1,\ \gamma = k,\ \Delta \ge 0,\ |\Delta| = c \text{ and } \kappa \cap \Delta = \{\} \quad \text{(hit)}, \\ 1 & \text{if } h = 0,\ \gamma = 0,\ \Delta \ge 0,\ |\Delta| = c \text{ and } \kappa \cap \Delta \neq \{\} \quad \text{(no hit)}, \\ 0 & \text{otherwise (impossible combination).} \end{cases}$$

There are $\binom{w-k}{c}$ sets $\Delta$ that are hits ($h = 1$, $\gamma = k$), where all changes are among the $w-k$ positions outside $\kappa$, and $\binom{w}{c} - \binom{w-k}{c}$ sets $\Delta$ that are non-hits ($h = 0$, $\gamma = 0$).

With analogous notation ($Y^0$, $Y^+$) as previously, we use $y := Y^0[\Delta, c, h, \gamma]$ to update *exactly* two values in $Y^+$ using the $\leftarrow_+$ operation:

1. Let $\Delta^+ := (\Delta \boxminus 1)$; we do not use a change at the new position.
   We update $Y^+[\Delta^+,\, c,\, h^+,\, \gamma + A_\Delta] \;\leftarrow_+\; y$, where $A_\Delta$ and $h^+$ are as in Section 3.4.
2. Let $\Delta^\dagger := (\Delta \boxminus 1) \cup \{w-1\}$; we use a change at the new position, so there can be no hit or newly covered positions. We update $Y^+[\Delta^\dagger,\, c+1, h, \gamma] \;\leftarrow_+\; y$.

As each count $y$ from $Y^0$ is added twice to counts in $Y^+$, the number of sequences doubles when the sequence length increases by 1, and since the sum over $Y[w, \ldots]$ is initially $\sum_{c=0}^{w} \binom{w}{c} = 2^w$, the invariant that the sum over $Y[i, \ldots]$ is $2^i$ is maintained for all $i \ge w$.

▷ Claim 5. The Count DP algorithm needs time $O(n^4 \cdot 2^{2w-1})$ and space $O(n^3 \cdot 2^{2w-1})$.

Proof. Each of the four $i, c, h, \gamma$ indices into $Y$ may take $O(n)$ distinct values; there are $2^{2w-1}$ sets $\Delta$ to consider, as $c$ is not bounded. Updates are constant-time per table value, and only the current $Y[i, \ldots]$ and $Y[i+1, \ldots]$ are in memory at the same time. ◁

## 3.6 Optimizations and Practical Considerations

We discuss improvements and practical alternatives to the basic DP algorithms.

**The counting DP and filtration efficiency curves.** Claim 5 shows that it is often impractical to use exact counting via dynamic programming to compute filtration efficiency curves, such as those in Figure 2. On the other hand, the filtration efficiencies do not need to be known exactly. Therefore, we may fall back to random sampling (with 10 million samples) for each value of $c$ to determine $\textsc{Filtered}_F(\kappa, n, c)$ up to 3 digits. Whenever $\binom{n}{c} \leq 10^7$, we can compute $\textsc{Filtered}_F(\kappa, n, c)$ almost instantly exactly by exhaustive enumeration of all sequences and computing the resulting number of $\kappa$-hits and covered positions.

There may also exist more efficient methods to compute $\textsc{Count}_F(\kappa, n, c)$ exactly, based on a more refined automaton-based analysis of the language of binary sequences to be enumerated [2], but the general problem of large state space appears unavoidable.

### 3.6.1 Engineering Optimizations

**Halving the sequence length.** We observe that the minimization DPs need not be run until $i = n$ but only up to approximately $n/2$. We discuss the optimization for the (simpler) MinHits case. For the MinCov case, the idea is similar, but we need a history of size of $2w - 1$ ($\Delta$ in Sections 3.3 and 3.4) instead of a size-$w$ history ($\Gamma$ in Section 3.2. For the MinCov|Hits case, we additionally need to track hits.)

Consider a history window $\Gamma$ of length $w$ placed near the middle of the sequence. Suppose this window contains $c' \leq c$ changes, and let $r := c - c'$ be the number of remaining changes that must be placed outside this window.

This middle window acts as a separator between the left and right parts of the sequence. Since each seed has span $w$, no seed can simultaneously depend on positions strictly to the left and strictly to the right of this window. Therefore, changes placed on the right side of the middle window cannot affect hits located on the left side, and changes placed on the left side cannot affect hits located on the right side. Hence, after fixing the configuration of changes $\Gamma$ inside the middle window, the two sides become independent subproblems.

Let $\ell_{\text{left}}$ and $\ell_{\text{right}}$ denote the lengths of the left and right parts of the sequence, respectively, such that $n = \ell_{\text{left}} + w + \ell_{\text{right}}$. The position of the middle window can be chosen such that the two sides are almost balanced, i.e., $|\ell_{\text{left}} - \ell_{\text{right}}| \leq 1$. Therefore, it is sufficient to compute the DP only up to length $\max\{\ell_{\text{left}}, \ell_{\text{right}}\} = \lceil (n - w)/2 \rceil$.

For each fixed placement of the $c'$ changes inside the middle window, the remaining $r$ changes are distributed between the two independent parts. More precisely, for each $r' = 0, 1, \ldots, r$, we place $r'$ changes on the left side and $r - r'$ changes on the right side. The corresponding MinHits value is then obtained by combining the DP values of the two subproblems. Finally, the minimum over all possible splits $r'$, and over all possible placements of the $c'$ changes inside the middle window, gives the desired value for this case.

**Reducing the number of changes in history sets by 1.** In the minimizing DPs above, the maximal number of changes to distribute across the length-$n$ sequence is $c$, and in principle, they may all be in the same history window of length $w$ (MinHits) or $2w - 1$ (MinCov, MinCov|Hits). We can special-case this particular situation and reduce the general DP from considering $c$ changes to only $c - 1$ changes, reducing the number of histories $\Gamma$ or $\Delta$ considerably.

If a particular history window contains all $c$ changes, then no additional changes remain to be placed elsewhere in the sequence. In the MinHits DP (Section 3.2), the update case 2 (new change) cannot happen anymore, and instead of looping through case 1 (no new change) until $i = n$, we pre-compute and tabulate the resulting MinHits values for all $\Gamma$ with $|\Gamma| = c$. Consequently, the resulting MinHits or MinCov value can be computed directly, without extending the dynamic programming recursion beyond this point.

We omit the technical details for MinCov and MinCov|Hits at this point.

▷ Claim 6. The modified MinHits DP algorithm needs time $O(nc \cdot \binom{w}{\leq c-1} + w\binom{w}{c})$ and space $O(c \cdot \binom{w}{\leq c})$.

Proof. This is a direct consequence of Claim 2 and the $c \mapsto c-1$ reduction. The $O(w\binom{w}{c})$ time term is for pre-computing the lookup table of size $\binom{w}{c}$. ◁

### 3.6.2 Asymptotic improvement: Limited changes per window

The minimum number of hits (or coverage) is typically not attained when all $c$ changes are concentrated within a one or a few history windows of length $w$ (or $2w-1$), particularly if $n \gg w$. In such cases, equivalent or better solutions usually exist in which the number of changes within any window of length $w$ is bounded by a value $u \leq c$. Under this assumption, the sets $\Gamma$ considered in the dynamic programming algorithm can be restricted to subsets of size at most $u$, rather than at most $c$. Consequently, the number of possible histories $\Gamma$ for the MinHits DP is reduced from $\binom{w}{\leq c}$ to $\binom{w}{\leq u}$, which may lead to a substantial reduction in both running time and memory usage. An analogous argument applies to the MinCov DPs.

The key question is how to find a safe upper bound $u$ that does guarantee to find the correct optimal MinHits value. We derive such a bound by analyzing the range over which a set of changes can affect seed hits.

Consider an infinite sequence $\ldots, s_{-1}, s_0, s_1, s_2, \ldots$ and a window $s_i, s_{i+1}, \ldots, s_{i+w-1}$ of length $w$. Any change within this window can only affect seeds that overlap it. The earliest affected seed starts at position $i - w + 1$, and the latest affected seed starts at position $i + w - 1$. Therefore, all affected seeds are contained within the interval $s_{i-w+1}, \ldots, s_{i+2w-2}$, which has length $3w - 2$.

Now let $u'$ denote the minimum number of changes required to destroy all hits in any sequence of length $3w - 1$. Suppose that a window of length $w$ contains $u'$ or more changes. Since these changes influence only a region of length at most $3w - 2$, but the $u'$ changes are sufficient to eliminate all hits in a region of size $3w - 1$, at least one of the changes inside the window is unnecessary and can be moved elsewhere (or omitted) without increasing the number of hits. It follows that there always exists an optimal solution in which every window of length $w$ contains at most $u := u' - 1$ changes.

This bound is useful when $u < c$, which may occur when the total number of changes is large. In the experiments presented in this paper, the number of changes is small, and thus this optimization has limited practical impact. Nevertheless, it provides an improvement and may be valuable for future studies involving larger values of $c$.

## 4 Experimental Results

### 4.1 Comparison of optimization times

We compare running times of the ILP and DP approach for the MinHits problem. Timings were run on an Ubuntu (24.04 LTS) server with two AMD EPYC 9534 64-Core Processors, 1.5 TB of 4800-MHz DDR5 memory, and a KIOXIA CD8P SSD. For ILPs, we use a free

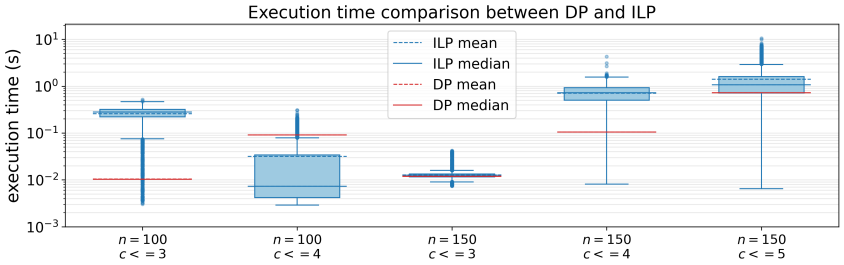



**Figure 3** Running times (logarithmic scale) for the ILP and DP approaches (without optimizations) for the MinHits objective on all masks of shape $(25, 35)$ with different sequence lengths $n$ and numbers of changes $c$. The blue boxplots and lines represent the distribution of ILP execution times over masks; the red lines indicate the corresponding (mask-independent) DP time statistics.

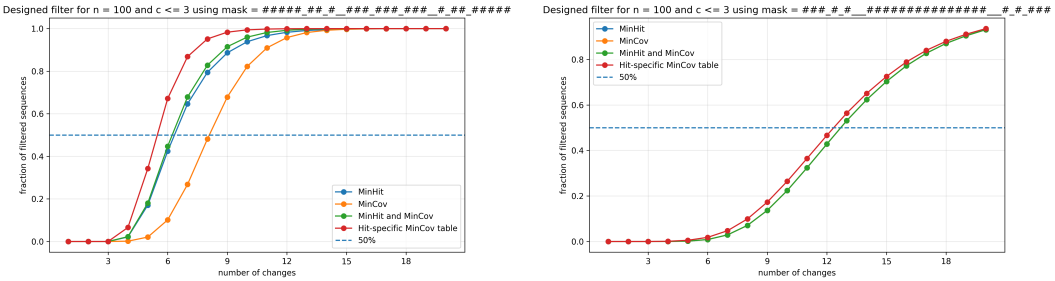



**Figure 4** Filtration efficiency curves for two masks of shape (25,35), with filtration criteria $F$ designed for $n = 100$ and $c^* = 3$ changes, for the four filter types introduced in Section 3.

academic Gurobi v13 license [7] and the Python API. Due to space constraints, we omit results for MinCov and MinCov|Hits, which are structurally similar but much longer overall. Figure 3 shows boxplots of the distribution of running times across all symmetric $(25, 35)$-shaped masks for different combinations of $n \in \{100, 150\}$ and $c \in \{3, 4, 5\}$. Since the DP timings are independent of the mask (the same work is done for each mask) and times may vary only due to possible uneven CPU or memory load on the server, only summary statistics are shown for DP. We observe that the ILP becomes faster for $n = 100$ when increasing $c$ from 3 to 4. This happens because for $c = 4$, there are many masks with MinHits=0 objective value, which is easy to find even by the LP relaxation. The DP running times increase exponentially with $c$, as expected. They appear to be independent of $n \in \{100, 150\}$, which is explained by the setup cost (table allocation, pre-computation) that is independent of $n$. For $n = 150$, the DP approach has an advantage over the ILP solver, which would become more pronounced when we increase $n$ and apply the proposed DP optimizations.

## 4.2 Comparison of masks

With both ILP and DP algorithms for MinHits, MinCov and MinCov|Hits, we can define the numerical criteria $F$ for the 4 filter types mentioned in Section 3 for any mask of interest, and using enumeration or sampling for computing $\text{Filtered}_F(\kappa, n, c)$ we can compute or approximate filtration efficiency curves.

As exemplary results, Figure 4 shows the four curves for the $(k, w) = (25, 35)$ masks `#####_##_#__###_###_###__#_##_#####` (left) and `###_#_#___###############___#_#_###` (right), with filtration criteria $F$ designed for $n = 100$ and $c^* = 3$ changes. Unless one

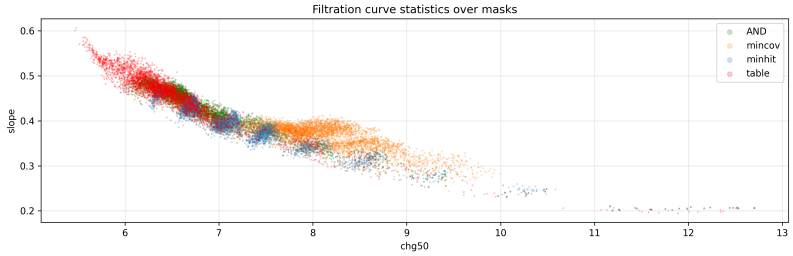



**Figure 5** Comparison of chg50 values and slopes of all four filter types (color) of all 4 368 symmetric masks of shape $(25, 35)$. Top left is best; bottom right is worst.

has different goals, one considers an exact filter $F$ better than another exact filter $F'$ if the curve of $F$ is above the curve of $F'$, because then $F$ filters out more of the undesired sequences with $c > c^*$ changes. (Both curves are at zero for $c \leq c^*$ because both $F$ and $F'$ are exact.) In the left panel of Figure 4, the MinCov filter (orange) performs worst, the MinHits (blue) filter better and close to the combined AND filter (green), which means that a simple logical AND of the hits and coverage criteria only yields a minor improvement, while the MinCov table filter (red; using MinCov|Hits, see Section 3) performs considerably better. This behavior is typical for many masks. However, for the mask in the right panel, the filtration curves for the MinHits, MinCov and combined AND filter agree (only the AND curve is therefore visible), the Table filter is only slightly better, and all four filters of this mask perform much worse than all filters of the left panel's mask.

We now introduce two quantities that capture the shape of the filtration curve and that may be used to compare the quality of (mask, filter) pairs designed for specific $n$ and $c^*$:

- the *chg50* value is defined as the number of changes where the (interpolated) curve crosses 50% filtration rate;
- the *slope* is defined as $\max_{c=0,1,2,\ldots}(f(c+2) - f(c))$, where $f$ is the filtration efficiency curve, the largest jump in filtration rate when the number of changes increases by 2. (The chosen difference of 2 is somewhat arbitrary, but better resolves between curves than a difference of 1.)

Figure 5 visually compares these two properties for all symmetric masks of shape $(25, 35)$. Several interesting observations may be made. The Table filters show the best filtration properties. The (red) point with largest slope ($> 0.6$) corresponds to the mask in the left panel of Figure 4; this can therefore considered to be the best mask. There is a number of badly performing masks (bottom right corner); and the mask in the right panel of Figure 4 is an example of them. On a side note, the MinHits filter chg50 values come in clusters on the chg50 axis, likely due to the discreteness of the low 1-digit MinHits numbers.

## 5 Discussion

We have made novel contributions to the field of spaced seed design: For the first time, we have designed a filtration strategy that combines hit-based and coverage-based criteria beyond a simple logical AND, and we have shown the table-based MinCov|Hits strategy to be quite effective for most masks. To compute the MinCov|Hits table, we have given both an ILP and a DP formulation that builds on simpler (but structurally similar) DPs for the related MinHits and MinCov optimization problems. The DP formulations reduce the number of change configurations to be considered from $\binom{n}{\leq c}$ to $\binom{w}{\leq c}$, independently of $n$. We have furthermore pointed out potential optimizations for reducing the running time of the

DP approaches. Implementing them and examining their effect for longer sequences is part of ongoing work. For the small number of changes that are of interest for short sequences ($n = 100$), we found them not to be relevant. However, the number of changes of interest grows with $\Theta(n)$ as the sequence length grows, and reducing the numbers of history sets from $\binom{w}{\leq c} = 2^w$ for $c \geq w$ to $\binom{w}{\leq u}$ for a small value of $u < w$ that does not depend on $c$ becomes crucial.

While we found the exact computation of filtration efficiency curves (for large $w$ or $c$) to be impractical, we analyzed the curves for all $(25, 35)$-shaped symmetric masks within a small case study by random sampling up to three digits accuracy. We defined two characterizing quantities of a mask/filter pair $(\kappa, F)$ designed to be exact for a given sequence length $n$ and number of changes $c^*$: the chg50 value and the slope of the filtration curve. We observed large differences between masks and found the optimal symmetric $(25, 35)$ mask for $n = 100$ and $c^* = 3$ to be `#####_##_#__###_###_###__#_##_#####`.

In ongoing and future work, we intend to build an atlas for spaced seeds with their hit- and coverage-related properties and filtration efficiencies, examine alternative bounds for limiting the number of changes per history window in the DP approaches, and extend the ideas presented here to combinations of masks. In addition, where this work only provides theoretical insights, the practical consequences of using an optimal vs. an average mask should be quantified in terms of wall-clock time speedup. Naturally, such an evaluation would depend on many additional design decisions and parameters.

We also need to point out a practical limitation of the approach: The MinCov|Hits thresholds assume that no additional random hits occur; otherwise, the filter may lose its exactness property. In practice, this may happen, depending on how the filter is used in a concrete application. For example, assume that all $\kappa$-mers of the human genome are stored in an index and the $\kappa$-mers of a sequenced DNA read of length $n = 100$ are queried. Assume further that the read is indeed a substring of a (unique) location in the genome with $c = 3$ substitutions that occur at certain positions of the read. This defines a hit pattern and coverage pattern on the read, but additionally, some of the changed $\kappa$-mers may occur *elsewhere* in the genome, yielding additional hits and additional covered positions, if the index only stores the presence of the $\kappa$-mer in the genome, but not its position(s). A problem can occur if the resulting hit and coverage pattern are not among the possible $\binom{n}{\leq c}$ patterns for at most $c$ changes; the observed coverage may then be *below* the MinCov|Hits value of the increased hit value because it was not considered in the optimization. Here is an example with a $(25, 35)$-mask, $c = 3$, $n = 100$, and 12 hits (with MinCov|Hits 77) plus one additional random hit, yielding a coverage of 77 plus 1, while the MinCov|Hits for 13 hits from 3 changes is 79:

```
mask:      #####_##_#__###_###_###__#_##_#####  (shape (25, 35))
3 changes: 00000000000000000000000000000000001000000000000000000000000000011000000000000000000000000000000000
12+1 hits: 0000010010110+0100010001101001000000000001000000000000100000000000  (+ is an additional random hit)
77+1 cov:  00000111111111111111111111111111111+1111111111111111111111111111100111111111111001011011111000000000000
```

The problem can be completely avoided by computing safe MinCov thresholds that are valid for *all* hit distributions with $h$ hits, but this reduces filtration efficiency.

Code is available at `https://gitlab.com/rahmannlab/seed-optimization`.

## References


1 B. Brejová, D. G. Brown, and T. Vinar. Optimal spaced seeds for homologous coding regions. *J Bioinform Comput Biol*, 1(4):595–610, January 2004. doi:10.1142/S0219720004000326.

2 Karel Brinda. Languages of lossless seeds. In Zoltán Ésik and Zoltán Fülöp, editors, *Proceedings 14th International Conference on Automata and Formal Languages, AFL 2014, Szeged, Hungary, May 27-29, 2014*, volume 151 of *EPTCS*, pages 139–150, 2014. doi:10.4204/EPTCS.151.9.

3 Stefan Burkhardt, Andreas Crauser, Paolo Ferragina, Hans-Peter Lenhof, Eric Rivals, and Martin Vingron. $q$-gram based database searching using a suffix array (QUASAR). In Sorin Istrail, Pavel A. Pevzner, and Michael S. Waterman, editors, *Proceedings of the Third Annual International Conference on Research in Computational Molecular Biology, RECOMB 1999, Lyon, France, April 11-14, 1999*, pages 77–83. ACM, 1999. doi:10.1145/299432.299460.

4 Stefan Burkhardt and Juha Kärkkäinen. Better filtering with gapped q-grams. *Fundam. Informaticae*, 56(1-2):51–70, 2003. URL: http://content.iospress.com/articles/fundamenta-informaticae/fi56-1-2-04.

5 K. Břinda, M. Sykulski, and G. Kucherov. Spaced seeds improve k-mer-based metagenomic classification. *Bioinformatics*, 31(22):3584–3592, November 2015. doi:10.1093/BIOINFORMATICS/BTV419.

6 M. C. Frith and L. Noé. Improved search heuristics find 20,000 new alignments between human and mouse genomes. *Nucleic Acids Res*, 42(7):e59, April 2014.

7 Gurobi Optimization, LLC. Gurobi Optimizer reference manual, 2023. URL: https://www.gurobi.com.

8 L. Hahn, C. A. Leimeister, R. Ounit, S. Lonardi, and B. Morgenstern. rasbhari: Optimizing spaced seeds for database searching, read mapping and alignment-free sequence comparison. *PLoS Comput Biol*, 12(10):e1005107, October 2016. doi:10.1371/JOURNAL.PCBI.1005107.

9 Moein Karami, Jens Zentgraf, and Sven Rahmann. Spaced Seed Optimization. Software, version 0.11., swhId: swh:1:dir:050c8caccda68021f2feb9b7b8400444e9888340 (visited on 2026-08-14). URL: https://gitlab.com/rahmannlab/seed-optimization/, doi:10.4230/artifacts.27677.

10 Denise Y. F. Mak and Gary Benson. All hits all the time: parameter-free calculation of spaced seed sensitivity. *Bioinformics*, 25(3):302–308, 2009. doi:10.1093/BIOINFORMATICS/BTN643.

11 François Nicolas and Eric Rivals. Hardness of optimal spaced seed design. *J. Comput. Syst. Sci.*, 74(5):831–849, 2008. doi:10.1016/J.JCSS.2007.10.001.

12 Laurent Noé and Donald E. K. Martin. A coverage criterion for spaced seeds and its applications to support vector machine string kernels and $k$-mer distances. *J. Comput. Biol.*, 21(12):947–963, 2014. doi:10.1089/CMB.2014.0173.

13 L. Noé. Best hits of 11110110111: model-free selection and parameter-free sensitivity calculation of spaced seeds. *Algorithms Mol Biol*, 12:1, 2017.

14 Kim R. Rasmussen, Jens Stoye, and Eugene W. Myers. Efficient q-gram filters for finding all epsilon-matches over a given length. In Satoru Miyano, Jill P. Mesirov, Simon Kasif, Sorin Istrail, Pavel A. Pevzner, and Michael S. Waterman, editors, *Research in Computational Molecular Biology, 9th Annual International Conference, RECOMB 2005, Cambridge, MA, USA, May 14-18, 2005, Proceedings*, Lecture Notes in Computer Science, pages 189–203. Springer, 2005. doi:10.1007/11415770_15.

15 Jens Zentgraf and Sven Rahmann. Design of worst-case-optimal spaced seeds. In Brona Brejová and Rob Patro, editors, *25th International Conference on Algorithms for Bioinformatics, WABI 2025, University of Maryland, College Park, MD, USA, August 20-22, 2025*, LIPIcs, pages 22:1–22:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.WABI.2025.22.

16 L. Zhang. Superiority of spaced seeds for homology search. *IEEE/ACM Trans Comput Biol Bioinform*, 4(3):496–505, 2007. doi:10.1109/TCBB.2007.1013.

# The Anti-Lexicographic SUS-Anchor: An Empirically Optimal Selection Scheme

**Ragnar Groot Koerkamp**
Karlsruhe Institute of Technology, Germany

## Abstract

**Motivation.** *Selection schemes* provide a way to select a subset of positions in a text in such a way that no two consecutive selected positions are more than $w$ apart. These selected positions can be used as "anchor" points for text indices such that every sufficiently long pattern corresponds to at least one anchor [2]. Closely related are *sampling schemes*, that sample a *$k$-mer* from each window of $w$ consecutive $k$-mers in a text, and the more restricted *minimizer schemes*, that achieve this by taking the smallest $k$-mer according to some order. In recent years, there has been a renewed interest in the search for *low density* schemes that select/sample only a small fraction of positions/$k$-mers.

The mod-minimizer [18] provides a near-optimal density of $1/w$ as $k/w \to \infty$, while schemes such as the greedy minimizer work well for explicit small parameters roughly in the regime $k \leq 2w$, for $k$ and $w$ up to 15 or so.

When $k < \log_\sigma w$ is small, minimizer schemes cannot do well [27]. As a first step towards low density sampling schemes in this regime, we fix $k = 1$ and search for a near-optimal selection scheme to improve the existing bidirectional string anchors (bd-anchors) [26, 2].

**Methods.** Inspired by bd-anchors, we introduce the *smallest unique substring* or SUS-anchor: given a window, this considers all suffixes that do not occur as a substring elsewhere in the window. It then samples the start position of the smallest suffix according to the new *anti-lexicographic* order that minimizes the first character and maximizes the remaining characters. We give a linear-time and $O(w)$ space streaming algorithm to compute all SUS-anchors of a string.

**Results.** For alphabet size $\sigma = 4$ and $k = 1$, the parameter-free anti-lexicographic SUS-anchor empirically has density $< 1\%$ away from the density lower bound and is at least $4\times$ closer to the lower bound than all other tested schemes. For alphabet size $\sigma = 2$, the density is at most 10% above the lower bound, which still improves $2\times$ to $3\times$ the overhead of the random minimizer and is consistently better than the greedy minimizer. Likewise, the anti-lexicographic minimizer performs better than all other schemes apart from the greedy minimizer.

**2012 ACM Subject Classification** Theory of computation → Sketching and sampling; Applied computing → Bioinformatics

**Keywords and phrases** Minimizers, Sampling scheme, Sketching, Maximal suffix, Smallest unique substring

**Digital Object Identifier** 10.4230/LIPIcs.WABI.2026.22

**Supplementary Material** *Software (Source Code)*: `https://github.com/RagnarGrootKoerkamp/minimizers` [14], archived at `swh:1:dir:6b775d1315385c94ecbd6058cfbba2476fbeb0b4`

**Acknowledgements** I thank the anonymous reviewers for their helpful feedback.

## 1 Introduction

Minimizers [36, 35] are a technique to subsample the *$k$-mers* of a string such that consecutive samples are at most $w$ positions apart. Minimizer sampling has many applications in bioinformatics. In particular, a number of data structures exploit this *window guarantee* and use minimizers as a locality-sensitive hash: SSHash [32, 33] and *k*ache-hash [22] use the minimizer of a string as the key in a hash map, while the U-index [1] is a text index that considers the minimizers of a pattern. In a similar way, the text index of Loukides et al.


© Ragnar Groot Koerkamp;
licensed under Creative Commons License CC-BY 4.0

and Ayad et al. [26, 2] works for patterns of length at least $\ell$ and selects *positions* in the text (*anchors*) that are at most $w = \ell$ apart in a locally consistent way, so that each queried pattern of length $\geq \ell$ can be found via its anchor. Improving their bidirectional string anchors is the main goal of this work, as this directly leads to a smaller text index.

From another point of view, the minimizers of a string are a *sketch* or compressed (lossy) representation, allowing for faster processing. Specific applications include seeding for read-mapping as done by minimap [25], the minimizer-space De Bruijn graph [10], sketching for Jaccard similarity in mashmap [21], host depletion in Deacon [7], and more [41]. In this direction, there are also other schemes such as syncmers [9] and universal hitting sets [29, 30, 28]. In these schemes, each $k$-mer is sampled independently from its context, which may loosen the window guarantee and does not directly give a suitable key or anchor for each window.

**Recent work.** There is a long line of active work on minimizers. Of particular interest is the *density*, which is the expected fraction of sampled positions. Recently, the mod-minimizer [18, 16] introduced schemes with near-optimal density for $k \to \infty$. The GreedyMini [13] is the state of the art for smaller $w$ and $k$ roughly up to 15, achieving particularly close to optimal density for $k = w + 1$, $k = w$, and $k$ just below $w$. Shur et al. recently introduced *spacers* [38] that prefer sampling $k$-mers starting with `10` (after projecting to a binary alphabet) for which the next occurrence of `10` is as far ahead as possible. Spacers improve the density of the ABB+ scheme we introduce here, and are especially good when $w$ is large. For particularly small parameters $(\sigma, k) = (2, \leq 6)$ and $(\sigma, k) = (4, 2)$, exactly optimal *minimizer* schemes are given by the search algorithm OptMini [39] and analysed theoretically by Shur [37].

This raises the question: **can we find near-optimal *sampling* schemes for small $k$, and in particular for $k = 1$**? An ILP search suggests that the answer is yes: this is always possible when $k \equiv 1 \mod w$ [23], and thus we are motivated to search for a "clean", constant-space scheme.

Some further recent work includes SimdMinimizers [17], a SIMD-based algorithm that computes *random minimizers* at around 500 Mbp/s. A precise analysis of the density of the random minimizer is given in [12]. Vigemers [19] reduce the imbalance when using minimizers for partitioning, and multiminimizers [20] reduce the density at the cost of more compute.

**Sampling and selection schemes.** Minimizer schemes hash each $k$-mer and sample the $k$-mer with the smallest hash in a window of $w$ $k$-mers (or $\ell = w + k - 1$ characters). Minimizers cannot achieve a good density of $O(1/w)$ for constant $k$ as $w \to \infty$ [27], but this restriction does not apply to *sampling schemes*, which have more freedom because they are not required to first hash all $k$-mers. In particular, *selection schemes* with $k = 1$ simply sample a single *position* based on a window of length $w = \ell$. The main reason that selection schemes can achieve better density than sampling schemes is that for a fixed sized window, they can also sample the last $k - 1$ positions, and thus achieve a greater distance between consecutive samples. Bidirectional string anchors (bd-anchors) are one such selection scheme [26]. These sample the start position of the smallest *rotation* of a window and achieve a density of $O(1/w)$. For large $w$ and alphabet size $\sigma = 4$, bd-anchors have a density around 20% above the lower bound. Our goal is to reduce this to as close to 0% overhead as possible.

**Contributions.** In this paper, we introduce SUS-anchors, short for *smallest unique substring* anchors. As the name implies, given a window, these sample the start position of the smallest substring that does not occur a second time. The *lexicographic* SUS-anchor is equivalent (after reversing the order of the alphabet) to the the *maximal suffix* of each window. Our main results are as follows:

- We experimentally show that SUS-anchors with the anti-lexicographic order (see below) have density within 1% of optimal for $\sigma = 4$ and any $w$.
- We give an $O(n)$ time and $O(\ell)$ space sliding-window algorithm to compute the SUS-anchors of a string of length $n$ that is based on the *monotone-queue* approach [17].

Smaller contributions that are of possible independent interest are:

- *character-based orders* that generalize e.g. the lexicographic, alternating [35], and ABB [11] order; and
- two new character-based orders: the *ABB+* and *anti-lexicographic* order.

Parts of these results have been previously introduced in the author's PhD thesis [15].

# 2 Preliminaries

## 2.1 Minimizer, sampling, and selection schemes

For a more in-depth introduction to minimizers, we refer the reader to the survey by Zheng et al. [41], the mod-minimizer paper [18], and the author's thesis [15]. Here we briefly introduce the required notation and concepts.

**Notation.** For an integer $w$, we write $[w] = \{0, 1, \ldots, w-1\}$, and for a string $W = w_0 \ldots w_{|W|-1}$ we use $W[i \ldots j)$ to indicate the substring $w_i \ldots w_{j-1}$. We assume an alphabet $\Sigma$ of size $\sigma$.

**Sampling schemes.** Given parameters $w \geq 1$ and $k \geq 1$, a *window* is a string over $\Sigma$ of length $\ell = w + k - 1$ that contains exactly $w$ $k$-mers. A *local sampling scheme* or just *sampling scheme* is a function $f : \Sigma^\ell \to [w]$ that indicates that from window $W$, the $k$-mer $W[f(W) \ldots f(W) + k)$ is *sampled*.

Given a text $T$, we are interested in the *set* of all sampled positions when sliding window $W$ over the text. Consecutive sampled positions differ by at most $w$, and a sampling scheme is *forward* when the sampled position in $T$ never decreases when sliding the window forward.

Often considered are *minimizer schemes* [36, 35], which are forward sampling schemes that sample (the position of) the smallest $k$-mer in the window according to some order as given by a hash function.

**Selection schemes.** In this paper, we are particularly interested in *selection* schemes, which are sampling schemes with $k = 1$ [40]. In particular, these select a *position* rather than *$k$-mer* in each window. Given a fixed window (context) of $\ell$ characters, the $w = \ell$ selection scheme has greater freedom than an $w = \ell - k + 1$ sampling scheme, and thus can achieve lower density.

**Density.** The *particular density* of a sampling scheme is the fraction of positions that are sampled. The *density* is the expected value of the particular density on a random string with length going to infinity. For forward schemes, the density can be computed as the probability that two *consecutive* windows $W$ and $W'$ (overlapping by $\ell - 1$ characters and together forming a *context* of $\ell + 1$ characters) sample a different position.

**Density lower bounds.** A trivial lower bound on the density of sampling schemes is $1/w$, since at least one in every $w$ positions is sampled. The best current lower bound for *forward* sampling schemes [23] simplifies (with a small loss of accuracy) to $\frac{1}{w+k}\left\lceil\frac{w+k}{w}\right\rceil$. For $k=1$, this gives a lower bound for forward selection schemes of $\frac{1}{w+1}\left\lceil\frac{w+1}{w}\right\rceil = \frac{2}{w+1}$. The main insight is that the density equals the expected number of samples in a random cyclic string (cycle) of length $w+1$, and in any such cycle, at least two different positions must be sampled.

For *minimizer* schemes, a lower bound is given by $1/\sigma^k$ [27]: if $w \to \infty$, exactly all occurrences of the smallest $k$-mer will be sampled, and these occur every $\sigma^k$ positions in expectation. Note, however, that this bound does *not* hold for general sampling schemes, and it is exactly this property that will allow us to design near-optimal sampling schemes for small $k$.

## 2.2 Bidirectional string anchors

*Bidirectional string anchors* (*bd-anchors*) are a $k=1$ selection scheme introduced by Loukides, Pissis, and Sweering for the purpose of text indexing [26]. Given a window, they sample the (leftmost) start position of the lexicographically smallest rotation. A drawback of bd-anchors is that they are not forward: the window `ZABAAC` has `AAC...` as smallest rotation, while the shifted window `ABAACA` has `AAB...` as smallest rotation. After further shifting to `BAACAY`, the smallest rotation is `AAC...` again: the `AB` prefix caused the selected position to jump around. To ensure the density[1] is in $O(1/w)$, *reduced* bd-anchors [26] avoid sampling the last $r = C\lceil\log_\sigma \ell\rceil$ positions, where $C=4$ in theory but in practice $C=3$ or less suffices. Given the choice of $r > \log_\sigma \ell$, in most windows of a random string the bd-anchor is the same as the smallest lexicographic $(r+1)$-mer. Note that reduced bd-anchors are still not forward.

In practice, bd-anchors can be computed in $O(n\ell)$ time by using Booth's linear-time algorithm for the lexicographically minimal rotation [6]. An $O(n)$ time algorithm is possible using a data structure of Kociumaka [24], but this is mostly of theoretical interest only since it requires $O(n)$ words of space for e.g. a suffix array. Theorem 3 of [26] gives an $O(n)$ time and $O(\ell)$ space algorithm when the alphabet size is constant ($\sigma = O(1)$) by using Kociumaka's method on overlapping chunks of size e.g. $2\ell$. A practical linear time algorithm to compute reduced bd-anchors with $r = \lceil 4\log_\sigma w\rceil$ is given by [2]. This works by first computing the lexicographic minimizers of size $k=r$ and only comparing the corresponding rotations in case of ties. Additionally, the *randomized* reduced bd-anchor is introduced that mimics the random minimizer: it considers the smallest $r$-mer in a window under a pseudo-random order and breaks ties by preferring the lexicographically smallest rotation. In practice, the density of randomized reduced bd-anchors is indistinguishable from the density of random minimizers.

## 2.3 Maximal suffixes

Separate from the literature on minimizers, there is a line of work on the *non-empty minimal suffix* and *maximal suffix* of a string [8]. While these two problems appear similar, as one might simply reverse the order of the alphabet, a crucial point is that a string $A$ is always smaller than $B$ when $A$ is a prefix of $B$. Thus, the minimal suffix prefers shorter suffixes, while the maximal suffix prefers longer suffixes. Because of this, minimal suffixes are less stable as we slide a window over a text, and we do not further consider them.

[1] Note that even though typically $\ell = w+k-1 = w$, we will use $w$ for the density and $\ell$ for the length of the window.

Babenko et al. [4] introduce an algorithm that preprocesses a string of length $n$ into a linear-space data structure and can then find the maximal suffix of query substrings of length $\ell$ in $O(\log \ell)$ time, which was improved to $O(1)$ query time in [3].

## 3 Character-based orders

To capture some of the existing "lexicographic-like" orders, we define *character-based orders* that compare strings one character at a time[2]. This is a natural requirement in our setting, because it allows comparing suffixes of a string without worrying that future characters (after shifting the window) will change the order, as long as one is not a prefix of the other.

▶ **Definition 1** (Character-based order). *An order $O$ on strings over $\Sigma$ is* character-based *if there exist orders $O_i$ on $\Sigma$ for $i \in \{0, 1, 2, \dots\}$ such that for all strings $A = a_0 \dots a_{|A|-1}$ and $B = b_0 \dots b_{|B|-1}$ with longest common prefix $\ell = \mathsf{lcp}(A, B)$ we have*

$$A <_O B \qquad \text{iff} \qquad (A \text{ is a strict prefix of } B) \text{ or } a_\ell <_{O_\ell} b_\ell.$$

*The orders can be either* total orders*, or* linear preorders *when equalities are allowed.*

This scheme encapsulates the **lexicographic order** on strings, where each $O_i$ is simply the lexicographic order on $\Sigma$. The main drawback of the lexicographic order is that it clusters small strings: since `AAAAX` is small, the next $k$-mer `AAAXY` is also small, possibly causing consecutive positions to be sampled as minimizers. Most of the following schemes instead look for a *transition* from a small character (`A`) to a large (`Z`) or non-small (`BCD..Z`) character.

The **alternating order** [35] uses the lexicographic order for even $i$, and the reverse lexicographic order for odd $i$, so that `AZAZAZ...` is the smallest string. The **ABB order** [5, 11] uses the lexicographic order for $O_0$, and for $i > 0$ it uses the order

$$1 =_{O_i} 2 =_{O_i} \cdots =_{O_i} \sigma - 1 <_{O_i} 0,$$

so that any string like `ABBBB...` or `AXYZDEF...` is minimal. This scheme has the nice property that occurrences of small strings starting in `A` and not containing further `A`'s are disjoint [5]. (The concept of sets of minimally overlapping strings has been further explored in [11], but these do not form a character-based order and do not directly work with variable-length strings.) **Vigemers** [19] are also a character-based order, where $O_i$ is the order after xor'ing by a character $\gamma_i$.

A drawback of the ABB order is that it throws away some information: for example, over the normal alphabet, `AB` and `AC` are considered equal. Thus, we also consider a version with tiebreaking, *ABB+*:

▶ **Definition 2** (ABB+ order). *The* ABB+ *order first compares two strings via the ABB order, and, in case of a tie, compares them via the plain lexicographic order.*

Note that this is only useful when comparing strings ($k$-mers) of equal length, and that the ABB+ order is not itself a character-based order.

The following scheme is more practical.

▶ **Definition 3** (Anti-lexicographic order). *The* anti-lexicographic order *uses the lexicographic order for $O_0$, and* reverse *lexicographic order for $O_i$ for $i > 0$.*

In this order, the smallest string is `AZZZZ...`.

[2] A slight generalization of this concept that we do not otherwise need in this paper uses orders $O_i$ on strings of length $i$ and then compares *prefixes* $a_0 \dots a_\ell <_{O_\ell} b_0 \dots b_\ell$ instead.

## 4 Smallest unique substring anchors

**Intuition.** Consider again the bd-anchor, which samples the start position of the smallest rotation of a window $W$. As we saw in Section 2.2, a drawback is that rotations "wrap around", and that the character $W[0]$ at the start of a string influences whether the rotation starting at the last character $W[\ell-1]$ is small or not. This is easily fixed by considering only the smallest *suffix* instead. However, the smallest suffix is the empty suffix, and so we could sample the first character of the smallest *non-empty* suffix. Unfortunately, this results in a bad density: a random window ends in the smallest symbol 0 with probability $1/\sigma$, in which case the suffix consisting of just this character is the smallest one, and the density will be around $1/\sigma$ regardless of $w$. To avoid this, we impose the following restriction (which, in a way, generalizes the "non-empty" condition): a suffix is only allowed to be sampled if it does not occur elsewhere in the window as a substring. Thus, we look for the *smallest unique suffix* $W[i\dots)$. Let $S = W[i\dots j)$ be the shortest prefix of $W[i\dots)$ that is unique in $W$. Then $S$ is the *smallest unique substring* (SUS[3]) of $W$ [15]. As an example, the string `CABBAB` has `AB` as its smallest suffix, and `ABBAB` as its smallest *unique* suffix. The smallest unique *substring* is `ABB`, and the index of the correspondingly sampled *SUS-anchor* is 1.

▶ **Definition 4** (SUS-anchor). *Given a window $W$ of length $w = \ell$, the* smallest unique substring *is the smallest substring $\mathsf{SUS}(W) = W[i\dots j)$ that does not occur elsewhere in $W$. Then $W[i\dots\ell)$ is the smallest unique* suffix*, and the SUS-anchor is $i$.*

We note that for the purposes of the definition, it would be sufficient to only consider the starting position of smallest unique *suffix*. Nevertheless, we prefer to consider the smallest unique *substring* since it has properties that are similar to the $k$-mers sampled by minimizer schemes. For example, both optimal-density minimizer schemes and optimal-density SUS-anchors would always sample non-overlapping $k$-mers and SUS-anchors respectively.

**Variants.** Alongside the lexicographic variant, the SUS-anchor allows variants based on character-based orders. Specifically, we consider SUS-anchors with the anti-lexicographic order. These variants work just like the lexicographic version: simply consider the set of suffixes that do not occur as a substring elsewhere, and then take the smallest of these using the chosen character-based order.

**MS-anchor.** It turns out that the concept of *lexicographic smallest unique suffix* is exactly equivalent to that of the *maximal suffix* [8, 3] after reversing the order of the alphabet: in both cases, we look for the extremal suffix where longer suffixes should be preferred over shorter ones. More generally, for any character-based order $O$, the SUS-anchor under $O$ starts at the same position as the maximal suffix when suffixes are compared by the reverse of $O$. Whereas the SUS-anchor explicitly requires considering a unique suffix, this is handled implicitly by the maximal suffix: when one suffix is a prefix of a longer suffix, the longer suffix is automatically the larger one (regardless of the chosen character-based order $O$).

▶ **Definition 5** (MS-anchor). *Given a window $W$ of length $w = \ell$, the* maximal suffix anchor *or MS-anchor samples the position $i \in [w]$ where the maximal suffix $W[i\dots\ell)$ of $W$ starts.*

[3] Not to be confused with the *shortest* unique substring, which is also commonly abbreviated as SUS.

## 4.1 Properties

We now state some observations and then prove some properties of SUS-anchors.

▶ **Observation 6.** *Given a window $W$, the smallest unique suffix is smaller than all longer suffixes, because all longer suffixes must be unique and thus larger than the smallest unique suffix.*

▶ **Observation 7.** *Removing the last character from the smallest unique substring results in a non-unique substring.*

▶ **Theorem 8** (SUS-anchors are forward)**.** *SUS-anchors are forward for any character-based order.*

**Proof.** Consider two consecutive windows $W$ and $W'$ with $\mathsf{SUS}(W) = W[i \dots j)$. If $i = 0$, the scheme is trivially forward. Otherwise, let $0 \leq i' < i - 1$ be the start index of a suffix $W'[i' \dots)$. By the previous observation, $W[i \dots) <_O W[i' + 1 \dots)$, and since $W[i \dots)$ is not a prefix of $W[i' + 1 \dots)$ (for otherwise it would not be unique), appending $W'[\ell - 1]$ to these two suffixes will not change their relative order. Since $W[i \dots)$ is unique in $W$, $W'[i - 1 \dots)$ is also unique in $W'$, and so $W'[i - 1 \dots)$ is a smaller unique suffix than $W'[i' \dots)$ for all $i' < i - 1$. Thus, $\mathsf{SUS}(W')$ cannot start at an index less than $i - 1$, and thus the SUS-anchor is forward. ◀

▶ **Theorem 9** (Charged context)**.** *Given two consecutive windows $W$ and $W'$ that together form a* context*, the sampled SUS-anchor changes if and only if either $\mathsf{SUS}(W)$ is a prefix of $W$ or $\mathsf{SUS}(W')$ is a suffix of $W'$. In this case, the context is* charged*.*

**Proof.** Let $\mathsf{SUS}(W) = W[i \dots j)$ and $\mathsf{SUS}(W') = W'[i' \dots j')$.

If $i = 0$ is sampled, $W'$ cannot sample the same position. If $W'[i' \dots j') = W[i' \dots \ell)$ is a suffix, then $W'[i' \dots j' - 1)$ is not unique in $W'$, and thus also not in $W$. Thus, $W$ must sample a different position.

For the other direction, assume that $i' > i - 1$, so that different text positions are sampled. If $i = 0$ or $i' = \ell - 1$ we are done, so assume $i > 0$ and $i' < \ell - 1$.

If $W[i \dots) < W[i' + 1 \dots)$, then for every possible character $W'[\ell - 1]$ that can be appended, we have $W'[i - 1 \dots) < W'[i' \dots)$, which is in contradiction with the fact that $W'[i' \dots)$ is smaller than all longer suffixes. Thus, $W[i' + 1 \dots)$ must be a prefix of $W[i \dots)$, so that the suffix is not unique. This means that $W'[i' \dots \ell - 1)$ is not unique in $W'$, and thus, the SUS $W'[i' \dots j')$ can only end in $j' = \ell$, as required. ◀

## 4.2 Computing SUS-anchors

Via the equivalence to maximal suffixes, we can compute all lexicographic SUS-anchors using the theoretical algorithm of [3] that requires $O(n)$ space, $O(n)$ preprocessing, and supports $O(1)$ queries for substring suffix-maximum. When the alphabet has constant size, we can use the same trick as for bd-anchors to reduce the space usage: Run the construction algorithm on chunks of size $2\ell$ that overlap by $\ell - 1$, so that the total space usage is $O(\ell)$ and the total time remains $O(n)$.

Below, we first adapt the method of [3] for computing the maximal suffix of each window to our streaming setting. Then, we adapt it to compute SUS-anchors for any character-based order.

▶ **Theorem 10.** *For any character-based order $O$, the SUS-anchors of a string of length $n$ over a constant-size alphabet can be computed in $O(n)$ time and $O(\ell)$ space.*

**A streaming algorithm for the maximal suffix.** We first adapt the algorithm of [3] to our streaming setting. Suppose that we have so far processed $T[0 \dots t)$. The algorithm maintains a doubly linked list $A = (a_0, a_1, a_2, \dots)$ of *active* text positions $a_i$ such that $T[a_i \dots t) > T[j \dots t)$ for all $a_i < j < t$. This list is *decreasing* in the sense that $T[a_0 \dots t) > T[a_1 \dots t) > T[a_2 \dots t) > \dots$, and it plays a similar role to the *monotone queue* in the classic minimizer algorithm [17]. Unlike that original case, here we also store a list of *events* for every text position that can modify the middle of the list.

As we shift the window and increment $t$ to $t' = t + 1$, we remove $a_0$ when $a_0 < t' - \ell$. After appending $T[t]$, we get a new suffix $T[t \dots t')$. We update $A$ before pushing the new element $t$.

- **Repeatedly pop $A$.** We remove the last element $\mathsf{last}(A) = A_{|A|-1}$ from $A$ as long as $T[\mathsf{last}(A) \dots t') < T[t \dots t')$.
- **Add event.** Then, if $A$ is not empty and $T[\mathsf{last}(A) \dots) < T[t \dots)$ (taking into account upcoming characters), compute $\lambda = \mathsf{LCP}(T[t \dots), T[\mathsf{last}(A) \dots))$ and add the event "check if we should drop the element preceding $t$ from $A$" to the event queue for position $t + \lambda + 1$.
- **Push $t$.** Then, push $t$ to $A$ and increment $t$ to $t' = t + 1$.
- **Handle event.** As soon as $t''$ reaches position $t + \lambda + 1$, we check if $t$ is still in $A$, and if so, if it has now become larger than its preceding element. Then repeatedly remove the preceding element as long as $T[t \dots t'')$ is larger, and end when either there is no preceding element, or when $T[a_i \dots t'') > T[t \dots t'')$. In that case, again check if the order will flip in the future, and if so, add a new event to the event queue.

**Practical considerations.** To make this practical, all $\mathsf{LCP}$ computations can be bounded to $\ell$, and the event queues can be stored in a ring buffer with just $\ell + 1$ slots for the current and next $\ell$ positions. Originally, the $\mathsf{LCP}$ computations are done in constant time using the suffix array. Instead, we simply do a word-by-word scan on the bit-representation of the text, which takes expected constant time on random strings since LCPs of, say, over 64 bits are unlikely in random strings.

**A more direct algorithm for SUS-anchors.** We now make some modifications to use the algorithm above for SUS-anchors. Most importantly, $A$ will now be an *increasing* list of suffixes, with the smallest at the start, with the caveat that unlike usually a string is considered *smaller* than all its prefixes. A final modification drops the event queues and allows us to replace the doubly linked list $A$ by a simple double-ended queue, as shown in Algorithm 1. For each $a \in A$, we now additionally store the text position $x_a$ where $a$ becomes "hot", i.e., we store that $a$ becomes the first element of $A$ (and thus the start of the maximal suffix of the window) when the character $T[x_a]$ enters the sliding window. When comparing $t$ to $\mathsf{last}(A)$, there are three cases.

1. $A$ is empty, in which case we push $t$ and set $x_t = t$.
2. $T[\mathsf{last}(A) \dots \mathsf{last}(A) + \ell) \leq_O T[t \dots t + \ell)$, in which case we push $t$ and note that it becomes hot (the smallest suffix) when $\mathsf{last}(A)$ falls out of the window, i.e., $x_t = \mathsf{last}(A) + \ell$.
3. $T[\mathsf{last}(A) \dots \mathsf{last}(A) + \ell) >_O T[t \dots t + \ell)$: $t$ "takes over" $\mathsf{last}(A)$ when character $t + \lambda$ enters the window, where $\lambda = \mathsf{LCP}(T[t \dots), T[\mathsf{last}(A) \dots))$.
   - If $\mathsf{last}(A)$ is already hot by that time ($x_{\mathsf{last}(A)} < t + \lambda$), push $t$ with $x_t = \min(t + \lambda, a + \ell)$.
   - If not, $\mathsf{last}(A)$ will never be hot. We pop it and (repeatedly) compare $t$ to $\mathsf{last}(A)$.

Once this process finishes, the first element of $A$ indicates the maximum suffix of the current window. We then increment $t$ to the next text position and check whether we reached the position $x_{a_1}$ where the next element of $A$ becomes hot (either because $a_0$ falls out of the window or $a_1$ introduces a new maximal suffix). In that case, we pop $a_0$.

**Algorithm 1** Pseudo code for the streaming algorithm to compute the SUS-anchors as given in Section 4.2.

```
1:  procedure SUS-ANCHORS(T, ℓ, O)
2:      A ← []                                  ▷ Deque of (position, hot-time) pairs.
3:      S ← []                                  ▷ List of SUS-anchor positions of each window.
4:      for t ← 0, 1, ..., n − 1 do
5:          x_t ← t                             ▷ Moment where T[t...) becomes hot.
6:          while A is not empty do
7:              (a, x_a) ← last(A)
8:              if T[a...a+ℓ) ≤_O T[t...t+ℓ) then      ▷ New suffix is larger.
9:                  x_t ← a + ℓ                 ▷ Hot when a exits the window.
10:                 break
11:             else                            ▷ New suffix is smaller: takes over a at time x
12:                 λ ← LCP(T[t...t+ℓ), T[a...a+ℓ))
13:                 x ← min(t + λ, a + ℓ)
14:                 if x_a < x then             ▷ a becomes hot before t takes over; push t after a.
15:                     x_t ← x
16:                     break
17:                 else                        ▷ a will never be hot; discard it and continue.
18:                     A.pop_last()
19:         A.push((t, x_t))
20:         (p, x_p) ← A.front()
21:         S.push(p)                           ▷ SUS-anchor of T[t − ℓ + 1...t + 1)
22:         if p + ℓ − 1 = t or (|A| > 1 and A.index(1).x ≤ t + 1) then
23:             A.pop_front()                   ▷ Pop from A if window ends or next element becomes hot.
24:     return S
```

## 5 Results

In Figure 1, we compare the density of SUS-anchors against the bd-anchors selection scheme as well as minimizer schemes. For the minimizer schemes, we choose the $k$ that minimizes the density, and we adjust $w$ to $w' = w + 1 - k$ accordingly to preserve the overall window length $\ell = w' + k - 1$. The true density is approximated by computing the *particular density* on a uniform random string of $10^7$ characters over alphabets of size 2, 4, and 32. A new independent random string is used for each data point.

Lexicographic minimizers are roughly on-par with bd-anchors, and lexicographic SUS-anchors are slightly better but still far from optimal. Double decycling [31] and the random minimizer appear to converge to density factor 2 as $w \to \infty$ and have similar density. The greedy minimizer [13] only performs well for small $w$ and $\sigma$, as it requires precomputed orders that are not always available. (We did not implement the logic to extrapolate from smaller orders.) The greedy minimizer is the only scheme that improves over anti-lexicographic minimizers (excluding very small $w \leq 6$).

The anti-lexicographic minimizer consistently improves over the random minimizer and double decycling. The anti-lexicographic SUS-anchors consistently have the best density of all, while also being parameter-free. They get surprisingly close to the lower bound: they are away from the lower bound by less than 10% for $\sigma = 2$ and less than 1% for $\sigma = 4$ and $\sigma = 32$. For $\sigma = 32$, the anti-lexicographic sus-anchor is indistinguishable from optimal for all $w \leq 64$, and improves over the random minimizer that is $> 1\%$ away from optimal here.

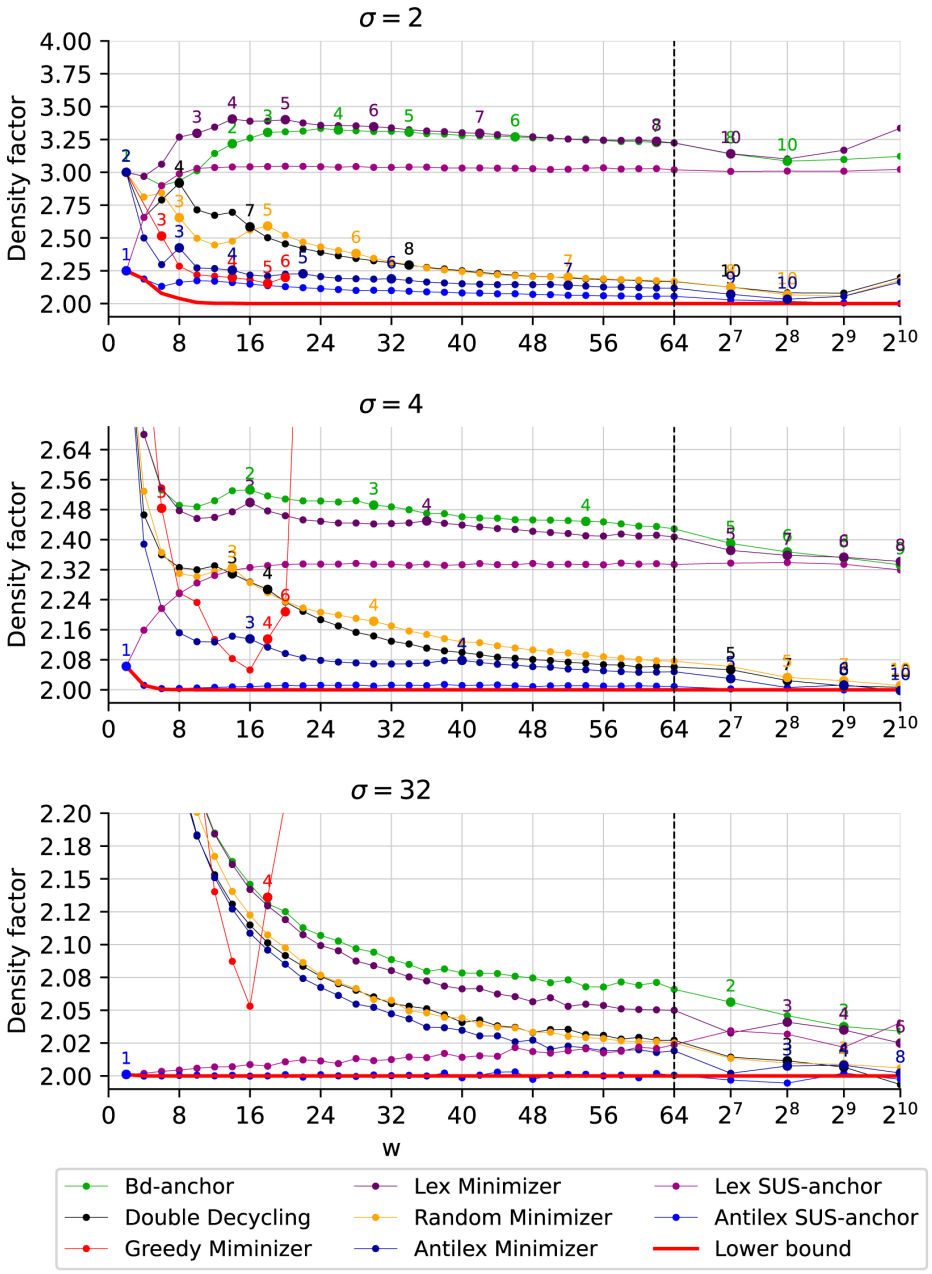



**Figure 1** Comparison of the density factor (the density multiplied by $w+1$) of selection schemes for $\sigma \in \{2, 4, 32\}$, and varying $w$. The lower bound is shown in red. For each scheme (minimizers, bd-anchors [26], double decycling [31], greedy minimizer [13]), the best parameter $r$ or $k$ is chosen for each $w$ (and $w$ is decreased accordingly). Changes to the value of $r$ or $k$ are annotated. On the right of each plot, we show scaling for large $w$. Each data point is the particular density on an independent random string of length $10^7$. For large $\sigma$ and $w$, there remains some variance in the estimated density, occasionally leading to estimates below the lower bound. Sus-anchors are consistently better than minimizers, and the anti-lexicographic order is better than the random order which itself is better than the lexicographic order.

**Table 1** Density factor and throughput of various selection schemes for $w = 31$ (left half) and $w = 63$ (right half) on a random string and on chromosome 19. For minimizer schemes, the value of $k$ that minimizes the density on chromosome 19 is chosen. Schemes are sorted by decreasing density.

| | $w = 31$ | Density | | Thrpt. | $w = 63$ | Density | | Thrpt. |
|---|---|---|---|---|---|---|---|---|
| Scheme | $k$ | Random | Ch.19 | Mbp/s | $k$ | Random | Ch.19 | Mbp/s |
| Bd-anchor | 2 | 2.59 | 3.04 | 2.0 | 3 | 2.53 | 3.36 | 1.0 |
| Greedy minimizer | 4 | 3.62 | 2.75 | 44.9 | 5 | 3.68 | 2.86 | 43.9 |
| Lex minimizer | 4 | 2.47 | 2.91 | 35.3 | 5 | 2.42 | 3.16 | 35.4 |
| Lex SUS-anchor | – | 2.33 | 2.66 | 19.0 | – | 2.34 | 2.96 | 18.4 |
| Random minimizer | 4 | 2.18 | 2.27 | 40.3 | 4 | 2.08 | 2.19 | 40.7 |
| Double decycling | 5 | 2.18 | 2.23 | 13.6 | 5 | 2.10 | 2.13 | 13.7 |
| Antilex minimizer | 3 | 2.07 | 2.29 | 36.2 | 4 | 2.05 | 2.42 | 35.0 |
| Antilex SUS-anchor | – | 2.01 | 2.18 | 17.9 | – | 2.01 | 2.33 | 17.6 |

**Density and throughput on chromosome 19.** In Table 1, we compare the density on a random string with the particular density on chromosome 19 of CHM13 [34], which has length 61.7 Mbp. Most schemes have slightly worse density on chromosome 19 than on real data. The greedy minimizer has worse than expected performance in both cases, which is likely due to the fact that our parameters are large and require extrapolating from smaller cases. The anti-lexicographic SUS-anchor has the best performance on random data, but on chromosome 19, it is outperformed by the random minimizer and double decyling [31] for $w = 63$.

Additionally, Table 1 shows the obtained throughput of evaluating each scheme on an Intel i7-10750H CPU running at 2.6 GHz, but we note that the implementations have not been optimized for speed. In particular, for bd-anchors we use a very naive $O(nw^2)$ implementation. Most other methods are $O(n)$ in practice, as can be seen from the fact that they do not slow down from $w = 31$ to $w = 63$. SIMD-minimizers [17], an optimized library for computing random minimizers, has a throughput over 500 MB/s, which is more than 10× faster than all of the methods shown here.

**Intuition on SUS-anchor density.** Low-density schemes sample positions that are as far apart as possible. When sampling $k$-mers, this means that ideally each $k$-mer does not overlap with the preceding or succeeding sampled $k$-mer, or else as little as possible.[4] Thus, shorter SUS-anchors are beneficial since they are both less likely to overlap and can be the anchor for more consecutive $k$-mers.

Taken together, this means that short unique substrings should work well. In particular, the anti-lexicographic order is better than the lexicographic order since it avoids sampling overlapping substrings: in the lexicographic order, consecutive substrings like `AAAB` and `AAB` are both small, causing overlapping samples when a window starts with such a string. The anti-lexicographic order prevents this by explicitly preferring a large suffix after removing the first character. A weakness of the ABB scheme is that it needs to sample a relatively

[4] In particular, a minimizer scheme that has density exactly equal to the lower bound [23] must never sample overlapping $k$-mers: The lower bound uses that at least two $k$-mers must be sampled on each primitive cycle of $w + k$ characters, and if two overlapping $k$-mers are sampled, there must be at least a third sampled $k$-mer. Similarly, SUS-anchors can only have optimal density if they never overlap.

long substring before it becomes unique: in a window like ABDDCBA, 6 of the 7 characters (ABDDCB) are part of the ABB-SUS, so that this sample can only cover a few windows. The anti-lexicographic SUS is simply AB instead.

This also explains the effect of the alphabet size: for $\sigma = 2$, on average a substring needs to have length $\log_2 w$ before it becomes unique. Thus, the SUS-anchors are longer and more likely to overlap. An example of a case where anti-lexicographic SUS-anchors still overlap is given by the consecutive windows ABABABB and BABABBB, with anti-lexicographic SUS-anchors ABABA and ABA. For alphabet size $\sigma = 2$, such cases are somewhat common. For $\sigma = 32$ on the other hand, most characters in the window are unique (for $w \leq 32$) and most anchors will consist of only one or two characters (e.g. just A or AZ) and thus rarely overlap.

## 6 Conclusion

We introduced anti-lexicographic SUS-anchors, gave a linear-time algorithm to compute them, and showed that they empirically get to within 1% of the density lower bound for $\sigma = 4$. They are consistently the lowest density selection scheme of all tested variants, while also being parameter-free. As a second best, anti-lexicographic minimizers also perform better than all other schemes apart from the greedy minimizer.

**Future work.** More work is needed to *prove* that, for example, anti-lexicographic SUS anchors have a close-to-optimal density of $2/(w+1) + o(1/w)$. Given how close to optimal these schemes already are, the fact that *exactly* optimal schemes are known to exist for small $w$ [23], including manually constructed for $\sigma = 2$ and $w \leq 12$, the question remains whether "clean", constant space schemes can be developed for generic $w$. The *spacers* of [38] are similar to our own ongoing work, and provide a first step in this direction.

On a practical level, further work is needed to improve the current monotone-queue-based algorithm to a rescan-like approach [17] or even a branchless variant that could be implemented in parallel using SIMD.

It would also be interesting to use SUS-anchors in combination with the mod-minimizer [18]. In particular, the mod-minimizer requires a lower bound $r = \Theta(\log_\sigma w)$ on the length of the chosen minimizers (typically around 4 in practice), and it would be interesting to see if "mod-SUS-anchors" can work without such a restriction. More generally, SUS-anchors can be extended to $k > 1$ by simply excluding the last $k - 1$ suffixes, and preliminary results show that this gives good density up to $k \approx \log_\sigma w$.

Lastly, the SUS-anchor is a forward scheme. It is known that non-forward schemes can be (at least) slightly better at times and break the forward lower bound, but this is not well understood.

## References


1 Lorraine A. K. Ayad, Gabriele Fici, Ragnar Groot Koerkamp, Grigorios Loukides, Rob Patro, Giulio Ermanno Pibiri, and Solon P. Pissis. U-Index: A Universal Indexing Framework for Matching Long Patterns. In *SEA 2025*, volume 338 of *LIPIcs*, pages 4:1–4:18, 2025. doi:10.4230/LIPIcs.SEA.2025.4.

2 Lorraine A. K. Ayad, Grigorios Loukides, and Solon P. Pissis. Text indexing for long patterns using locally consistent anchors. *The VLDB Journal*, 34(5), July 2025. doi:10.1007/s00778-025-00935-7.

3 Maxim Babenko, Paweł Gawrychowski, Tomasz Kociumaka, Ignat Kolesnichenko, and Tatiana Starikovskaya. Computing minimal and maximal suffixes of a substring. *Theoretical Computer Science*, 638:112–121, 2016. Pattern Matching, Text Data Structures and Compression. doi:10.1016/j.tcs.2015.08.023.

4 Maxim Babenko, Ignat Kolesnichenko, and Tatiana Starikovskaya. On minimal and maximal suffixes of a substring. In *Combinatorial Pattern Matching*, pages 28–37. Springer Berlin Heidelberg, 2013. doi:10.1007/978-3-642-38905-4_5.

5 Simon R. Blackburn. Non-overlapping codes. *IEEE Transactions on Information Theory*, 61(9):4890–4894, September 2015. doi:10.1109/tit.2015.2456634.

6 Kellogg S. Booth. Lexicographically least circular substrings. *Information Processing Letters*, 10(4):240–242, 1980. doi:10.1016/0020-0190(80)90149-0.

7 Bede Constantinides, John Lees, and Derrick W Crook. Deacon: fast sequence filtering and contaminant depletion. *bioRxiv*, June 2025. doi:10.1101/2025.06.09.658732.

8 Jean-Pierre Duval. Factorizing words over an ordered alphabet. *Journal of Algorithms*, 4(4):363–381, 1983. doi:10.1016/0196-6774(83)90017-2.

9 Robert Edgar. Syncmers are more sensitive than minimizers for selecting conserved k-mers in biological sequences. *PeerJ*, 9:e10805, February 2021. doi:10.7717/peerj.10805.

10 Barış Ekim, Bonnie Berger, and Rayan Chikhi. Minimizer-space De Bruijn graphs: Whole-genome assembly of long reads in minutes on a personal computer. *Cell Systems*, 12(10):958–968.e6, October 2021. doi:10.1016/j.cels.2021.08.009.

11 Martin C Frith, Laurent Noé, and Gregory Kucherov. Minimally overlapping words for sequence similarity search. *Bioinformatics*, 36(22–23):5344–5350, December 2020. doi:10.1093/bioinformatics/btaa1054.

12 Shay Golan and Arseny M. Shur. Expected density of random minimizers. In *SOFSEM 2025: Theory and Practice of Computer Science*, pages 347–360. Springer Nature Switzerland, 2025. doi:10.1007/978-3-031-82670-2_25.

13 Shay Golan, Ido Tziony, Matan Kraus, Yaron Orenstein, and Arseny Shur. GreedyMini: generating low-density DNA minimizers. *Bioinformatics*, 41:275–284, July 2025. doi:10.1093/bioinformatics/btaf251.

14 Ragnar Groot Koerkamp. RagnarGrootKoerkamp/minimizers. Software, swhId: swh:1:dir:6b775d1315385c94ecbd6058cfbba2476fbeb0b4 (visited on 2026-08-14). URL: https://github.com/RagnarGrootKoerkamp/minimizers, doi:10.4230/artifacts.27679.

15 Ragnar Groot Koerkamp. *Optimal Throughput Bioinformatics*. PhD thesis, ETH Zurich, 2025. doi:10.3929/ETHZ-C-000783091.

16 Ragnar Groot Koerkamp, Daniel Liu, and Giulio Ermanno Pibiri. The open-closed mod-minimizer algorithm. *Algorithms for Molecular Biology*, 20(4), March 2025. doi:10.1186/s13015-025-00270-0.

17 Ragnar Groot Koerkamp and Igor Martayan. SimdMinimizers: Computing Random Minimizers, fast. In *SEA 2025*, volume 338 of *LIPIcs*, pages 20:1–20:19, 2025. doi:10.4230/LIPIcs.SEA.2025.20.

18 Ragnar Groot Koerkamp and Giulio Ermanno Pibiri. The mod-minimizer: A simple and efficient sampling algorithm for long $k$-mers. In *WABI 2024*, volume 312 of *LIPIcs*, pages 11:1–11:23, 2024. doi:10.4230/LIPIcs.WABI.2024.11.

19 Florian Ingels, Antoine Limasset, Camille Marchet, and Mikaël Salson. Vigemers: on the number of $k$-mers sharing the same xor-based minimizer. *arXiv*, 2026. doi:10.48550/arXiv.2602.03337.

20 Florian Ingels, Lucas Robidou, Igor Martayan, Camille Marchet, and Antoine Limasset. Minimizer density revisited: Models and multiminimizers. In *RECOMB*, November 2026. doi:10.1101/2025.11.21.689688.

21 Chirag Jain, Alexander Dilthey, Sergey Koren, Srinivas Aluru, and Adam M. Phillippy. A fast approximate algorithm for mapping long reads to large reference databases. *Journal of Computational Biology*, 25(7):766–779, July 2018. doi:10.1089/cmb.2018.0036.

22 Jamshed Khan, Rob Patro, and Prashant Pandey. $k$ache-hash: A dynamic, concurrent, and cache-efficient hash table for streaming $k$-mer operations. *openRxiv*, February 2026. doi:10.64898/2026.02.13.705625.

23 Bryce Kille, Ragnar Groot Koerkamp, Drake McAdams, Alan Liu, and Todd J Treangen. A near-tight lower bound on the density of forward sampling schemes. *Bioinformatics*, December 2024. doi:10.1093/bioinformatics/btae736.

24 Tomasz Kociumaka. Minimal Suffix and Rotation of a Substring in Optimal Time. In Roberto Grossi and Moshe Lewenstein, editors, *27th Annual Symposium on Combinatorial Pattern Matching (CPM 2016)*, volume 54 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 28:1–28:12, Dagstuhl, Germany, 2016. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CPM.2016.28.

25 Heng Li. Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. *Bioinformatics*, 32(14):2103–2110, March 2016. doi:10.1093/bioinformatics/btw152.

26 Grigorios Loukides, Solon P. Pissis, and Michelle Sweering. Bidirectional string anchors for improved text indexing and top-$k$ similarity search. *IEEE Transactions on Knowledge and Data Engineering*, 35(11):11093–11111, November 2023. doi:10.1109/tkde.2022.3231780.

27 Guillaume Marçais, Dan DeBlasio, and Carl Kingsford. Asymptotically optimal minimizers schemes. *Bioinformatics*, 34(13):i13–i22, June 2018. doi:10.1093/bioinformatics/bty258.

28 Guillaume Marçais, David Pellow, Daniel Bork, Yaron Orenstein, Ron Shamir, and Carl Kingsford. Improving the performance of minimizers and winnowing schemes. *Bioinformatics*, 33(14):i110–i117, July 2017. doi:10.1093/bioinformatics/btx235.

29 Yaron Orenstein, David Pellow, Guillaume Marçais, Ron Shamir, and Carl Kingsford. Compact universal k-mer hitting sets. In *Algorithms in Bioinformatics*, pages 257–268. Springer International Publishing, 2016. doi:10.1007/978-3-319-43681-4_21.

30 Yaron Orenstein, David Pellow, Guillaume Marçais, Ron Shamir, and Carl Kingsford. Designing small universal k-mer hitting sets for improved analysis of high-throughput sequencing. *PLOS Computational Biology*, 13(10):e1005777, October 2017. doi:10.1371/journal.pcbi.1005777.

31 David Pellow, Lianrong Pu, Bariş Ekim, Lior Kotlar, Bonnie Berger, Ron Shamir, and Yaron Orenstein. Efficient minimizer orders for large values of $k$ using minimum decycling sets. *Genome Research*, August 2023. doi:10.1101/gr.277644.123.

32 Giulio Ermanno Pibiri. Sparse and skew hashing of $k$-mers. *Bioinformatics*, 38:i185–i194, June 2022. doi:10.1093/bioinformatics/btac245.

33 Giulio Ermanno Pibiri and Rob Patro. Optimizing sparse and skew hashing: faster $k$-mer dictionaries. *Bioinformatics*, 42, July 2026. doi:10.1093/bioinformatics/btag264.

34 Arang Rhie, Sergey Nurk, Monika Cechova, Savannah J. Hoyt, Dylan J. Taylor, Nicolas Altemose, Paul W. Hook, Sergey Koren, Mikko Rautiainen, Ivan A. Alexandrov, Jamie Allen, Mobin Asri, Andrey V. Bzikadze, Nae-Chyun Chen, Chen-Shan Chin, Mark Diekhans, Paul Flicek, Giulio Formenti, Arkarachai Fungtammasan, Carlos Garcia Giron, Erik Garrison, Ariel Gershman, Jennifer L. Gerton, Patrick G. S. Grady, Andrea Guarracino, Leanne Haggerty, Reza Halabian, Nancy F. Hansen, Robert Harris, Gabrielle A. Hartley, William T. Harvey, Marina Haukness, Jakob Heinz, Thibaut Hourlier, Robert M. Hubley, Sarah E. Hunt, Stephen Hwang, Miten Jain, Rupesh K. Kesharwani, Alexandra P. Lewis, Heng Li, Glennis A. Logsdon, Julian K. Lucas, Wojciech Makalowski, Christopher Markovic, Fergal J. Martin, Ann M. Mc Cartney, Rajiv C. McCoy, Jennifer McDaniel, Brandy M. McNulty, Paul Medvedev, Alla Mikheenko, Katherine M. Munson, Terence D. Murphy, Hugh E. Olsen, Nathan D. Olson, Luis F. Paulin, David Porubsky, Tamara Potapova, Fedor Ryabov, Steven L. Salzberg, Michael E. G. Sauria, Fritz J. Sedlazeck, Kishwar Shafin, Valery A. Shepelev, Alaina Shumate, Jessica M. Storer, Likhitha Surapaneni, Angela M. Taravella Oill, Françoise Thibaud-Nissen, Winston Timp, Marta Tomaszkiewicz, Mitchell R. Vollger, Brian P. Walenz, Allison C. Watwood, Matthias H. Weissensteiner, Aaron M. Wenger, Melissa A. Wilson, Samantha Zarate, Yiming Zhu, Justin M. Zook, Evan E. Eichler, Rachel J. O'Neill, Michael C. Schatz, Karen H. Miga, Kateryna D. Makova, and Adam M. Phillippy. The complete sequence of a human Y chromosome. *Nature*, 621(7978):344–354, August 2023. doi:10.1038/s41586-023-06457-y.

**35** Michael Roberts, Wayne Hayes, Brian R. Hunt, Stephen M. Mount, and James A. Yorke. Reducing storage requirements for biological sequence comparison. *Bioinformatics*, 20(18):3363–3369, July 2004. doi:10.1093/bioinformatics/bth408.

**36** Saul Schleimer, Daniel S. Wilkerson, and Alex Aiken. Winnowing: local algorithms for document fingerprinting. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD/PODS03. ACM, June 2003. doi:10.1145/872757.872770.

**37** Arseny Shur. On minimizers of minimum density. *arXiv*, 2025. doi:10.48550/arXiv.2506.05277.

**38** Arseny Shur, Ido Tziony, and Yaron Orenstein. 10-minimizers: a promising class of constant-space minimizers. *bioRxiv*, 2026. doi:10.64898/2026.03.16.712052.

**39** Arseny Shur, Ido Tziony, and Yaron Orenstein. Generating minimum-density minimizers. *openRxiv*, January 2026. doi:10.64898/2026.01.25.701585.

**40** Hongyu Zheng, Carl Kingsford, and Guillaume Marçais. Lower density selection schemes via small universal hitting sets with short remaining path length. *Journal of Computational Biology*, 28(4):395–409, April 2021. doi:10.1089/cmb.2020.0432.

**41** Hongyu Zheng, Guillaume Marçais, and Carl Kingsford. Creating and using minimizer sketches in computational genomics. *Journal of Computational Biology*, 30(12):1251–1276, 2023. doi:10.1089/cmb.2023.0094.

# Statistical Inconsistency of Error-Correction Objectives for Perfect Phylogenies

**Gryte Satas**
Halvorsen Center for Computational Oncology, Memorial Sloan Kettering Cancer Center, New York, NY, USA

**Matthew A. Myers**
Halvorsen Center for Computational Oncology, Memorial Sloan Kettering Cancer Center, New York, NY, USA

**Sohrab P. Shah**
Halvorsen Center for Computational Oncology, Memorial Sloan Kettering Cancer Center, New York, NY, USA

## Abstract

The binary perfect phylogeny, in which each mutation arises exactly once on an evolutionary tree and is never lost, is a well-studied idealized phylogenetic model. When observed data has errors, a common approach to phylogeny inference is to seek a tree that minimizes the number of error corrections ("flips") needed to fit a perfect phylogeny. These objectives draw on an intuitive justification: minimizing implied errors should prefer the true tree in expectation.

We test this assumption using a generative model with independent errors and prove that error-correction objectives are statistically inconsistent for all positive error rates: in expectation, the minimum-cost tree need not be the true tree. Our proof is constructive and yields counterexamples involving any tree topology and any positive error rates, demonstrating the ubiquity of the phenomenon. The core problem is that tree topologies can explain observed mutation patterns with fewer errors than actually occurred, and differ in their ability to do so, introducing systematic bias. This mechanism is distinct from previously identified sources of inconsistency such as homoplasy or incomplete lineage sorting, since the error-free setting is trivially consistent for perfect phylogenies.

We investigate how often this failure may occur in practice. Simulations calibrated to error rates from single-cell sequencing data show that an incorrect tree is preferred over the true tree in a substantial fraction of cases (over 50% in some settings) with rates increasing with tree size. Moreover, winning trees are not random but share specific topological features. Notably, at error rates typical of single-cell sequencing data, trees with deeper, more imbalanced topologies are consistently favored over more balanced ones. These results demonstrate that inconsistency is not a theoretical edge case, and that understanding when and how it arises is important when interpreting results in practice.

**2012 ACM Subject Classification** Theory of computation → Design and analysis of algorithms; Applied computing → Computational genomics

**Keywords and phrases** Phylogenetics, Perfect Phylogeny, Phylogenetic Inconsistency, Single-Cell Sequencing, Cancer Evolution

**Digital Object Identifier** 10.4230/LIPIcs.WABI.2026.23

**Supplementary Material** *Software*: `https://github.com/shahcompbio/phylo_inconsistency` [40] archived at `swh:1:dir:4b04560c2d51f196ba4119f87d0f4f858d3783e6`

**Funding** This research was supported in part by NCI SPORE (P50 CA247749-01).

## 1 Introduction

The binary perfect phylogeny is a well studied model in phylogenetics in which each mutation arises exactly once on an evolutionary tree and is inherited by all descendants, with no losses or reversions [27]. This has also been referred to as the infinite-sites model, as it can be viewed as a limit of finite-sites models where the number of sites grows faster than the number


© Gryte Satas, Matthew A. Myers, and Sohrab P. Shah;
licensed under Creative Commons License CC-BY 4.0



of mutations, so that homoplasy and reversion become vanishingly unlikely [31]. Under this idealized model, each mutation imposes a hard compatibility constraint on the tree topology. As a result, the underlying tree topology is uniquely determined once sufficiently many informative mutations are observed, and can be found in linear time [1, 23]. In many applications, however, mutations are not observed directly but are measured through a noisy process. In single-cell sequencing data, for example, allelic dropout and limited coverage produce false negatives, while sample preparation, sequencing, and alignment artifacts introduce false positives [48]. In the presence of such errors, mutations no longer impose exact compatibility constraints, and the guarantees of the error-free perfect-phylogeny model no longer apply. This has led to a family of combinatorial formulations for phylogeny inference under measurement error [9]. These approaches seek an optimal tree that minimizes the number or weighted cost of entry-wise corrections ("flips") to make the observed data compatible with a perfect phylogeny. The simplest case where all flips have equal cost is well known as the minimum-flip perfect phylogeny problem or the minimum-flip consensus tree problem [9]. Prior work has focused on studying the computational complexity and behavior of the problems and developing algorithms that find or approximate minimum-cost solutions [2, 5, 6, 8, 9, 10]. Applied variants of these objectives have been widely used in cancer phylogenetics, for reconstructing tumor evolutionary histories from noisy single-cell mutation data (see Related Work in Section 1.1).

Error-correction approaches are implicitly motivated by an intuition that minimizing the number of implied observation errors should recover the true tree without bias, at least when noise itself is unbiased and sufficient data are observed. However, to our knowledge, this assumption has not been systematically examined; it remains unclear whether these objectives yield statistically consistent estimators of the true tree under reasonable generative models of evolution and observation noise. Statistical consistency is a property of an estimator that guarantees convergence to the true parameter (in this case, the true tree) as the amount of data increases [21, 47]. It has been a topic of longstanding interest in phylogenetics [18, 24, 44] (see Related Work in Section 1.1); however, perfect phylogeny inference has not been examined in this context, likely because consistency is trivial in the error-free setting. The perfect phylogeny model is in this sense a natural minimal test case: it strips away all evolutionary sources of ambiguity that are known to cause phylogenetic inconsistency (e.g., homoplasy, incomplete lineage sorting), leaving only the interaction between observation noise and the objective to explain any inconsistency that arises. While few applied methods use the basic perfect phylogeny error-correction objective in its pure form, most retain the same core structure of hard-assigning mutations to branches and selecting the tree that minimizes the resulting cost. Thus, any inconsistency that arises here will likely carry through to those settings as well.

In this work, we study error-correction objectives for perfect phylogeny reconstruction under a generative model with independent observation errors. We show that these objectives are not statistically consistent for any positive error rates, even when the error penalties are perfectly calibrated to the true error rates. That is, even with known error rates and unbiased errors, the minimum-flip objective need not favor the true tree in expectation. Moreover, the objectives can be positively misleading, converging to a specific incorrect tree with increasing support as the number of mutations increases. The source of this inconsistency is not algorithmic failure, limited data, or ambiguity in the evolutionary model, but a structural interaction between observation noise and phylogenetic topology under the objective. The objective seeks a tree that explains the observed data with the fewest, or lowest cost, corrections. Consequently, errors are attributed to spurious evolutionary structure whenever doing so reduces the total cost of flips required to achieve compatibility.

Crucially, this process is not uniform with respect to tree topology: trees vary in their ability to absorb different types of errors. Certain trees can systematically explain false positive or false negative errors with fewer corrections than the true tree, leading to lower objective values in expectation. We prove that inconsistency is ubiquitous in tree space: there exist counterexamples that violate consistency for at least any pair of trees that differ by one internal edge and for any positive error rates.

Which topology is systematically favored is determined by error rates, tree topology and branch lengths; a topology may have a systematic advantage over the true tree under one branch length distribution but not another. While the theoretical proofs rely on arbitrary branch length distributions, we also demonstrate that constraining edge lengths to be equal does not eliminate inconsistency. We next performed computational analyses to investigate how often inconsistency arises under biologically motivated edge length distributions and error rates typical of single-cell sequencing data. We find that inconsistency arises in a non-trivial fraction of tested cases, reaching over 50% in some settings, with rates appearing to increase with tree size. At these error rates, the bias has a consistent directional signature toward more caterpillar-like trees. These results demonstrate that inconsistency in perfect phylogeny error-correction inference can arise in practice under non-pathological conditions.

## 1.1 Related work

**Consistency of maximum parsimony in classical phylogenetics.** Maximum parsimony is one of the oldest and most widely used approaches to phylogenetic reconstruction [22], seeking the tree that minimizes the total number of changes implied along its branches: a parsimony criterion applied directly to evolutionary events. Perfect phylogeny error-correction objectives are related in spirit but the parsimony criterion is applied to the observation error process rather than the evolutionary process.

Consistency of parsimony-based phylogenetic inference has been extensively studied [18, 24, 26, 42]. Maximum parsimony is known to be statistically inconsistent in certain parameter regimes, converging to an incorrect topology with increasing confidence as data accumulate. The phenomenon was first demonstrated by Felsenstein [18], who showed that under certain branch length configurations, long-branch attraction causes maximum parsimony to favor incorrect topologies, producing the "Felsenstein zone." Subsequent work generalized inconsistency results to arbitrary numbers of taxa and $k$-state character models as well as demonstrated constrained regions of parameter space where parsimony is consistent [26, 42, 43, 45]. A second and distinct source of inconsistency arises under the multispecies coalescent, where various concatenation-based methods can converge to an incorrect species tree, because incomplete lineage sorting can create systematic discordance across loci [15, 14, 28, 29, 34, 36, 35, 46, 50].

In both settings, inconsistency is a consequence of the evolutionary process itself: homoplasy or lineage sorting produce systematic biases that the parsimony objective cannot overcome. The results we present here demonstrate a distinct mode of estimator failure: the perfect phylogeny assumption precludes homoplasy by construction, and a lack of recombination means that lineage sorting is not a factor.

**Applications in cancer phylogenetics.** The rooted perfect phylogeny model has been widely adopted in cancer genomics, as tumor evolution operates over a relatively short time frame that limits the accumulation of homoplasy, and the normal genome provides a known ancestral state [4]. With the rise of single-cell sequencing technologies, perfect phylogeny error-correction objectives have become a natural fit for inference from noisy single-cell mutation data. A range of methods have been developed in this setting, including

both combinatorial and probabilistic formulations [3, 7, 16, 25, 30, 37, 38, 49, 52] as well as extensions that relax the perfect phylogeny assumption (e.g., Dollo or finite sites models), incorporate additional data modalities such as copy number, or combine multiple types of data (e.g., combining bulk and single cell) [11, 17, 20, 32, 33, 39, 41, 51, 53].

# 2 Background

## 2.1 Perfect phylogenies

A *phylogenetic tree* is a rooted binary tree $T$ on $n$ leaves with edge set $E(T)$. The root $r$ has a single child connected by the trunk edge $e_r$; all other internal nodes have exactly two children. Each edge $e$ has an associated length $p_e > 0$, normalized so that $\sum_{e \in E(T)} p_e = 1$.

▶ Remark 1 (Trunk edge convention). Representing trees with a trunk edge is common and convenient in cancer phylogenetics for modeling mutations shared by all cells. Such trees can be reconciled with the standard definition of a full rooted binary tree with the addition of a dummy outgroup attached to the root that is treated as a noiseless ancestral reference.

Mutation data are represented as a binary matrix $X \in \{0,1\}^{n \times m}$, where rows correspond to leaves and columns to mutations. We assume that $X$ is generated under the binary *perfect phylogeny model* [23], where each mutation arises exactly once on the tree and is inherited by all descendants. Let $T^*$ represent the true tree from which $X$ was generated. Each mutation independently selects an edge $e \in E(T^*)$ with probability $p_e$. This defines the true assignment function $\Phi^* : \{1, \ldots, m\} \to E(T^*)$, where $\Phi^*(j)$ is the edge on which mutation $j$ arises. More generally, for any candidate tree $T$, an *assignment function* $\Phi : \{1, \ldots, m\} \to E(T)$ maps each mutation to an edge of $T$. For each edge $e \in E(T)$, let $D_e$ denote its *descendant set*, i.e. the set of leaves whose root-to-leaf path contain $e$. Under the perfect phylogeny model, an assignment $\Phi$ induces a binary mutation matrix $X^\Phi \in \{0,1\}^{n \times m}$ defined by $X^\Phi_{i,j} = \mathbf{1}\{i \in D_{\Phi(j)}\}$, where $\mathbf{1}\{\ldots\}$ denotes the indicator function. That is, all descendant leaves of $e$ have state 1 for mutation $j$, and all other leaves have state 0.

▶ **Definition 2** (Compatibility). *A mutation matrix $X$ is **compatible** with a tree $T$ if there exists an assignment $\Phi$ such that $X = X^\Phi$.*

Thus, under a perfect phylogeny model with strictly positive edge lengths, each observed mutation places a hard constraint on the tree inference. Provided that sufficiently many mutations are observed to place at least one mutation on every internal edge, the true tree $T^*$ is the unique tree compatible with the data [23].

**Observation model.** Given a true mutation matrix $X \in \{0,1\}^{n \times m}$, generated under a perfect phylogeny model, we assume that the observed matrix $\hat{X} \in \{0,1\}^{n \times m}$ is generated under an error model where each entry of $X$ is flipped independently with fixed probability. That is,

$$\Pr(\hat{X}_{i,j} = 0 \mid X_{i,j} = 1) = \alpha \quad \text{and} \quad \Pr(\hat{X}_{i,j} = 1 \mid X_{i,j} = 0) = \beta,$$

where $\alpha \in (0, 0.5)$ is the false negative rate and $\beta \in (0, 0.5)$ is the false positive rate.

## 2.2 Minimum flip-cost perfect phylogeny problem (MFCP)

Under the observation model described above, the guarantees of the noiseless setting no longer hold: the observed matrix $\hat{X}$ is not in general compatible with the true tree, and compatibility alone cannot guide inference. This has motivated a family of well-studied

combinatorial problems for phylogeny inference for data with errors. Rather than requiring exact compatibility, these problems seek to minimize the number or cost of entry-wise corrections ("flips") needed to make the observed data compatible with a perfect phylogeny.

For a candidate tree $T$ and an assignment function $\Phi : \{1, \ldots, m\} \to E(T)$, discrepancies between the observed matrix $\hat{X}$ and the induced matrix $X^{\Phi}$ take two forms: $0 \to 1$ flips (entries where $\hat{X}_{i,j} = 0$ but $X^{\Phi}_{i,j} = 1$) and $1 \to 0$ flips (entries where $\hat{X}_{i,j} = 1$ but $X^{\Phi}_{i,j} = 0$). Let $w_P, w_N > 0$ denote the costs of $0 \to 1$ and $1 \to 0$ flips, respectively. Note that the flip directions are defined relative to editing the observed matrix $\hat{X}$ into a matrix compatible with a candidate tree, and should not be confused with the directions of the underlying observation errors. Likewise, $w_P$ and $w_N$ are user-defined parameters of the inference objective, and should not be confused with the latent error rates $\alpha$ and $\beta$ in the generative model. While $w_P$ and $w_N$ can be set to reflect the relative costs of false positives and false negatives, we make no assumptions about their relationship in the theoretical analyses that follow.

For two binary vectors $\mathbf{x}, \hat{\mathbf{x}} \in \{0,1\}^n$, the flip cost is

$$\text{cost}(\mathbf{x}; \hat{\mathbf{x}}) = w_P \sum_i \mathbf{1}\{\hat{x}_i = 0,\, x_i = 1\} + w_N \sum_i \mathbf{1}\{\hat{x}_i = 1,\, x_i = 0\}.$$

For any edge $e \in E(T)$, let $\mathbf{x}_e \in \{0,1\}^n$ denote the mutation pattern induced by $e$, where the 1-set of $\mathbf{x}_e$ corresponds to the descendant set $D_e$. The *flip cost* of a tree $T$ with respect to an observed matrix $\hat{X}$ is defined as

$$C(T,\, \hat{X}) = \sum_{j=1}^{m} \min_{e \in E(T)} \text{cost}(\mathbf{x}_e;\, \hat{\mathbf{x}}_j) = \min_{\Phi} \sum_{j=1}^{m} \text{cost}(\mathbf{x}_{\Phi(j)};\, \hat{\mathbf{x}}_j), \tag{1}$$

where $\hat{\mathbf{x}}_j$ is the $j$-th column of $\hat{X}$, and the minimum is taken over all assignment functions $\Phi : \{1, \ldots, m\} \to E(T)$. Note that $C(T, \hat{X})$ is invariant under multiplying both $w_P$ and $w_N$ by the same positive constant, and as such there is only one degree of freedom in the choice of weights, which can be parameterized by the ratio $w_P/w_N$. We will continue to refer to them as two separate parameters, as it provides a more intuitive interpretation. Throughout, $w_P$ and $w_N$ are treated as fixed parameters of the objective and suppressed from the notation of $C$ and all quantities derived from it.

This yields the following computational problem:

▶ **Problem 3** (Minimum flip-cost perfect phylogeny problem (MFCP)). *Given an observed binary matrix $\hat{X} \in \{0,1\}^{n \times m}$ and weights $w_P, w_N > 0$, find a tree $T$ minimizing the flip cost $T \in \text{argmin}_{T'} C(T', \hat{X})$.*

Specific choices for weights $w_P$ and $w_N$ yield well-studied special cases: the minimum flip problem (MFP, $w_P = w_N = 1$), the insertion-only flip problem (IFP, $w_N \gg w_P$, so $1 \to 0$ flips are never used), and the deletion-only flip problem (DFP, $w_P \gg w_N$, so $0 \to 1$ flips are never used) [9].

▶ Remark 4 (Relationship to maximum likelihood inference). A natural response to inconsistency of a combinatorial objective is to switch to a maximum likelihood formulation. However, jointly maximizing likelihood over trees and mutation assignments (i.e., $\max_{T,\Phi} \Pr(\hat{X} \mid T, \Phi)$) under the observation model described above yields an objective that falls within the MFCP family, so the inconsistency results here apply directly. Specifically, when error rates are assumed to be uniform and independent, the log-likelihood decomposes into a sum of per-entry penalties, and maximizing it is equivalent to minimizing a weighted flip cost with weights $w_P = \log \frac{1-\beta}{\alpha}$ and $w_N = \log \frac{1-\alpha}{\beta}$ (see Appendix A.4 for details). Some approaches use

entry-specific likelihoods incorporating additional information such as read counts; these can be viewed as generalizations of the MFCP objective in which the i.i.d. case is a special case. As they still rely on hard-assigning mutations to edges and optimizing a sum of penalties, the inconsistency mechanism identified here likely carries through to this more general setting as well.

# 3 Inconsistency of the Minimum Flip-Cost Perfect Phylogeny Objective

We now ask whether the MFCP objective can recover the true tree in principle. Specifically, does the true tree uniquely minimize the expected flip cost induced by the generative process?

## 3.1 Inconsistency

Because the total flip cost is additive across mutations and the assignment of each mutation can be optimized independently, the expected flip cost over a full matrix $\hat{X} \in \{0,1\}^{n\times m}$ decomposes as a sum of independent per-mutation contributions as in Equation (1). It therefore suffices to analyze the expected cost of a single mutation. Let $\phi^*$ denote the true edge on which the mutation occurs, i.e., for mutation $j$, $\phi^* = \Phi^*(j)$. We denote the expected flip cost of a candidate tree $T$ under generative model $(T^*, \mathbf{p})$ by

$$\bar{C}(T \mid T^*, \mathbf{p}) = \mathbb{E}_{\phi^*|\mathbf{p}}\Big[\mathbb{E}_{\hat{\mathbf{x}}|\alpha,\beta,\phi^*}\big[C(T,\hat{\mathbf{x}})\big]\Big] = \sum_{e\in E(T^*)} p_e\, \mathbb{E}_{\hat{\mathbf{x}}|\alpha,\beta,\phi^*=e}[C(T,\hat{\mathbf{x}})],$$

where the expectation is over both the edge $\phi^*$ on which the mutation arises in $T^*$ (conditional on fixed edge lengths $\mathbf{p}$), and the observation noise generating $\hat{\mathbf{x}}$ (conditional on $\phi^*$ and the error probabilities $\alpha$ and $\beta$). In the following analyses we treat $\alpha$ and $\beta$ as fixed parameters of the generative model and, for simplicity, suppress them from the notation of $\bar{C}$ and all quantities derived from it.

▶ **Definition 5** (Consistency). *Consider a true tree $T^*$, edge length distribution $\mathbf{p}$ on $E(T^*)$, and error rates $\alpha, \beta$. The MFCP objective with weights $w_N, w_P$ is* consistent *if the true tree $T^*$ uniquely minimizes the expected cost:*

$$\bar{C}(T^* \mid T^*, \mathbf{p}) < \bar{C}(T \mid T^*, \mathbf{p}) \quad \textit{for all } T \neq T^*.$$

▶ Remark 6. This definition corresponds to Fisher consistency [21]. In our setting (where data are i.i.d., costs are bounded, and the tree space is finite) Fisher consistency is necessary and sufficient for asymptotic consistency, meaning the estimator would converge to the true tree as the number of mutations increases [47]. See Appendix A.1 for a detailed.

For our main result we will show that under the generative model defined in Section 2.1, consistency fails for all objectives in the MFCP family:

▶ **Theorem 7** (Inconsistency of MFCP). *For any weights $w_N, w_P > 0$ and any error probabilities $\alpha, \beta \in (0, 0.5)$, there exist a true tree $T^*$ and an edge length distribution $\mathbf{p}$ for which the MFCP objective is not consistent.*

### Intuition: structural absorption of errors

The key mechanism underlying inconsistency for this problem is a form of overfitting to observation noise. The MFCP objective will always assign mutations to whichever edge explains their errors most cheaply. As a result, the true number of observation errors is only

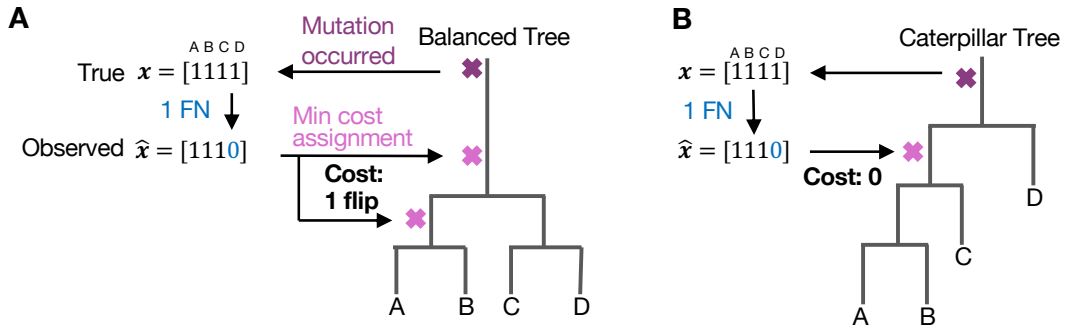



**Figure 1 Structural absorption of observation errors.** (**A**) A mutation arising on the trunk of a tree with balanced topology $((A, B), (C, D))$ has true pattern $\mathbf{x} = [1\,1\,1\,1]$, where the order of leaves is $A, B, C, D$. A false negative at leaf $D$ produces observed pattern $\hat{\mathbf{x}} = [1\,1\,1\,0]$, which is assigned to the minimum cost edge incurring a cost of one flip. (**B**) The same trunk mutation and false negative on the caterpillar tree $(((A, B), C), D)$ produces the same observed pattern $\hat{\mathbf{x}} = [1\,1\,1\,0]$. This observed pattern exactly matches the descendant set of the internal edge $\{A, B, C\}$. Therefore the minimum cost assignment for this pattern incurs zero cost, despite the fact that one observation error has occurred.

an upper bound on the overall cost: a tree can achieve lower cost by "absorbing" some errors, reinterpreting a noisy pattern as a clean signal from a different edge. Whether a tree can absorb a given error depends on its topology.

As an example, consider the trees shown in Figure 1: a balanced tree $((A, B), (C, D))$ and a caterpillar tree $(((A, B), C), D)$ on four leaves, with a mutation arising on the trunk. This mutation has true pattern $\mathbf{x} = [1\,1\,1\,1]$ (where the order of leaves is $A, B, C, D$). If a single false negative occurs on leaf $D$, this yields observed pattern $\hat{\mathbf{x}} = [1\,1\,1\,0]$. In the balanced tree the minimum cost assignment would require one flip (a $0 \to 1$ flip to the trunk $[1\,1\,1\,1]$, or a $1 \to 0$ flip to internal edge $[1\,1\,0\,0]$) concordant with the true number of errors (Fig. 1A). In contrast, in the caterpillar tree, the same false negative produces an observed pattern that exactly matches the descendant set of the internal edge $[1\,1\,1\,0]$ (Fig. 1B). This allows the mutation to be assigned to the tree with zero flips, despite the fact that one observation error has occurred. Conversely, the balanced tree can explain two simultaneous false negatives on $A$ and $B$ with zero flips, while the caterpillar would require one flip. However, when false negative rate $\alpha < 0.5$, single false negatives are more likely than double false negatives, so the caterpillar has a systematic advantage over the balanced tree for mutations arising on the trunk edge. We will show that these types of asymmetries are pervasive and unlikely to cancel out across edges.

## 3.2 Per-edge cost decomposition

We first develop the machinery required to analyze the difference in expected cost between the true tree $T^*$ and a candidate tree $T$. We decompose the expected cost difference $\Delta_{\mathbf{p}}(T, T^*)$ into per-edge contributions:

$$\Delta_{\mathbf{p}}(T, T^*) = \bar{C}(T \mid T^*, \mathbf{p}) - \bar{C}(T^* \mid T^*, \mathbf{p}) = \sum_{e \in E(T^*)} p_e \, \delta_e(T, T^*),$$

where $\delta_e(T, T^*)$ is the expected cost difference conditional on a mutation arising on edge $e$ of $T^*$:

$$\delta_e(T, T^*) = \mathbb{E}_{\hat{\mathbf{x}} \mid \phi^* = e, \alpha, \beta} \left[ C(T, \hat{\mathbf{x}}) - C(T^*, \hat{\mathbf{x}}) \right].$$

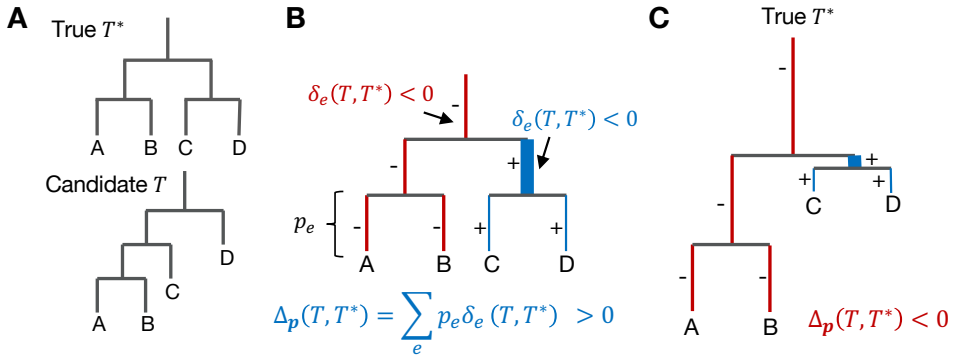



**Figure 2 Edge-level cost differences and the edge concentration argument.** (**A**) The true tree $T^*$ (balanced) and a candidate tree $T$ (caterpillar). (**B**) The true tree $T^*$ with a balanced edge length distribution. The expected cost difference $\Delta_{\mathbf{p}}(T, T^*)$ between the candidate tree $T$ and the true tree $T^*$ decomposes as a weighted sum of per-edge contributions $\delta_e(T, T^*)$. Edge color indicates the sign of $\delta_e(T, T^*)$ (red: negative, favoring the candidate; blue: positive, favoring the true tree), and edge width indicates magnitude. Under the balanced distribution, the weighted sum $\Delta_{\mathbf{p}}(T, T^*)$ is positive. (**C**) The true tree $T^*$ with a skewed edge length distribution that concentrates mass on the negatively-contributing edges. The sign and magnitude of $\delta_e(T, T^*)$ remain unchanged from (**B**). However, because more mass is placed on the negatively-contributing edges, the weighted sum becomes negative.

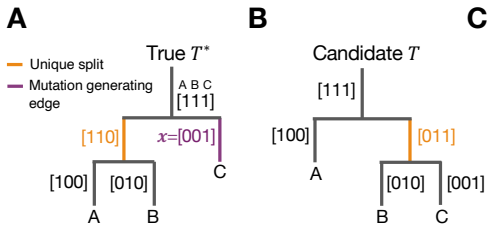
This decomposition allows us to analyze expected cost differences one generating edge at a time, separately from the edge length distribution. The following lemma shows that for consistency to hold, we would need $\delta_e(T, T^*) \geq 0$ for every $e \in E(T^*)$ and every $T \neq T^*$, as a single edge favoring a candidate tree is enough to break consistency (Fig. 2).

▶ **Lemma 8** (Edge concentration). *Consider a true tree $T^*$ and a candidate tree $T$. If there exists an edge $e \in E(T^*)$ such that $\delta_e(T, T^*) < 0$, then there exists an edge length distribution $\mathbf{p}$ under which $\bar{C}(T \mid T^*, \mathbf{p}) \leq \bar{C}(T^* \mid T^*, \mathbf{p})$.*

**Proof.** Let $\mathcal{S}^- = \{e' \in E(T^*) : \delta_{e'}(T, T^*) < 0\}$ be the set of edges favoring the candidate tree, and let $\mathcal{S}^+ = \{e' \in E(T^*) : \delta_{e'}(T, T^*) \geq 0\}$ be the set of edges favoring the true tree. The expected cost difference can be expressed as the sum of contributions from the two sets $\sum_{e' \in \mathcal{S}^-} p_{e'}\, \delta_{e'}(T, T^*)$ and $\sum_{e' \in \mathcal{S}^+} p_{e'}\, \delta_{e'}(T, T^*)$. Since $\mathcal{S}^-$ is non-empty, the sum can be made non-positive by choosing $\mathbf{p}$ to concentrate sufficient mass on edges in $\mathcal{S}^-$ and away from edges in $\mathcal{S}^+$. ◀

If $e$ is a shared split between two trees $T_1$ and $T_2$, then $\delta_e(T_1, T_2) = -\delta_e(T_2, T_1)$. As a result, showing a single shared edge $e$ such that $\delta_e(T_1, T_2) \neq 0$ for some pair of trees $T_1$ and $T_2$ is sufficient to conclude that consistency is violated, by Lemma 8.

Lemma 8 is the device we use to establish inconsistency in the proofs that follow, by demonstrating examples of trees and edges for which $\delta_e(T, T^*) < 0$. It assumes that edge lengths are unconstrained, so that mass can be redistributed toward the set $\mathcal{S}^-$ of negatively-contributing edges. This may raise concerns about whether violations of consistency require specially constructed or pathological edge length distributions. In Section 4, we will show that inconsistency does arise under reasonable edge length distributions and even for equal edge lengths.




| $\hat{x}$ | #FP | #FN | $\Pr(\hat{x} \mid x)$ | $C(T^*, \hat{x})$ | $C(T, \hat{x})$ |
|---|---|---|---|---|---|
| 000 | 0 | 1 | $(1-\beta)^2\alpha$ | 0 | 0 |
| 001 | 0 | 0 | $(1-\beta)^2(1-\alpha)$ | 0 | 0 |
| 010 | 1 | 1 | $(1-\beta)\beta\alpha$ | 0 | 0 |
| 100 | 1 | 1 | $\beta(1-\beta)\alpha$ | 0 | 0 |
| 011 | 1 | 0 | $(1-\beta)\beta(1-\alpha)$ | 1 flip | 0 |
| 101 | 1 | 0 | $\beta(1-\beta)(1-\alpha)$ | 1 | 1 |
| 110 | 2 | 1 | $\beta^2\alpha$ | 0 | 1 flip |
| 111 | 2 | 0 | $\beta^2(1-\alpha)$ | 1 | 1 |

**Figure 3 Three leaf violation of consistency (Lemma 9).** (**A**) True tree $T^*$ on leaves $A$, $B$, $C$, with $A$ and $B$ as siblings. The split [1 1 0] (orange) is unique to $T^*$, where the order of leaves is $A, B, C$. The mutation-generating edge (purple) is the leaf edge of $C$ with mutation pattern $\mathbf{x} = [0\,0\,1]$. (**B**) Candidate tree $T$ on the same leaf set, with $B$ and $C$ as siblings. The split [0 1 1] (orange) is unique to $T$. (**C**) All eight possible observed patterns $\hat{\mathbf{x}}$ given true mutation state $\mathbf{x} = [0\,0\,1]$, with the number of false positives (#FP) and false negatives (#FN), the probability $\Pr(\hat{\mathbf{x}} \mid \mathbf{x})$, and the minimum flip cost $C(T^*, \hat{\mathbf{x}})$ and $C(T, \hat{\mathbf{x}})$ on each tree. The only patterns for which the two trees differ in cost are $\hat{\mathbf{x}} = [1\,1\,0]$ and [0 1 1] (orange).

## 3.3 Inconsistency of MFCP: a three-leaf counterexample

We provide two complementary proofs. The first is a concrete three-leaf example where the calculations can be confirmed directly; the second is a general construction for any tree topology with $n \geq 3$ leaves, establishing that inconsistency is pervasive.

▶ **Lemma 9** (Counterexample with three leaves). *Let $T$ and $T^*$ be two distinct rooted binary trees on the same leaf set $\{A, B, C\}$. For any weights $w_N, w_P > 0$ and any error probabilities $\alpha, \beta \in (0, 0.5)$, there exists a probability distribution $\mathbf{p}$ such that $\bar{C}(T \mid T^*, \mathbf{p}) < \bar{C}(T^* \mid T^*, \mathbf{p})$.*

**Proof.** Without loss of generality, label the leaves on the trees so that $T^* = ((A, B), C)$ and $T = (A, (B, C))$, as shown in Figure 3A–B. We compute the expected cost difference $\delta_{e_c}(T, T^*)$ for the generating edge $e_c$ corresponding to the outgroup leaf $C$, where $\mathbf{x}_{e_c} = [0\,0\,1]$. As shown in Figure 3C, among all eight possible observed patterns, the only ones for which $T^*$ and $T$ differ in cost are the two tree-specific internal splits [1 1 0] and [0 1 1]: each has cost 0 on the tree that contains it and cost one flip (i.e., $\min(w_P, w_N)$) on the other. The expected cost difference $\delta_{e_c}(T, T^*)$ is therefore determined entirely by the relative probabilities of these two patterns: $\Pr(\hat{\mathbf{x}} = [0\,1\,1] \mid \mathbf{x} = [0\,0\,1]) = (1-\beta)\beta(1-\alpha)$ and $\Pr(\hat{\mathbf{x}} = [1\,1\,0] \mid \mathbf{x} = [0\,0\,1]) = \beta^2\alpha$. When $\alpha, \beta \in (0, 0.5)$, $(1-\beta)\beta(1-\alpha) > \beta^2\alpha$, so $\delta_{e_c}(T, T^*) < 0$. By Lemma 8, there exists a choice of edge lengths $\mathbf{p}$ such that $\bar{C}(T \mid T^*, \mathbf{p}) < \bar{C}(T^* \mid T^*, \mathbf{p})$. ◀

Given Lemma 9, we can now prove the main Theorem 7.

**Proof of Theorem 7.** For any weights and error probabilities, Lemma 9 provides a pair of distinct trees and an edge length distribution under which $\bar{C}(T \mid T^*, \mathbf{p}) < \bar{C}(T^* \mid T^*, \mathbf{p})$. Thus, by Definition 5, the MFCP objective is not consistent. ◀

## 3.4 Inconsistency of MFCP: a general construction for counterexamples

We now generalize the three-leaf counterexample to arbitrary tree topologies with $n \geq 3$ leaves. We use pairs of trees that differ in exactly one internal split. These can be constructed for any tree with $n \geq 3$ by performing a single rooted nearest-neighbor interchange (NNI) operation which swaps two subtrees across an internal edge [19]. Trees that differ by a single NNI are referred to as *NNI neighbors*. Any tree has two neighbors for every internal edge.

▶ **Lemma 10** (Counterexample with NNI neighbors of arbitrary size). *Let $T_1$ and $T_2$ be two distinct rooted binary trees on the same leaf set that are NNI neighbors. For any $w_P, w_N > 0$ and any $\alpha, \beta \in (0, 0.5)$, there exists an assignment of $T_1$ and $T_2$ to the roles of true tree $T^*$ and candidate tree $T$, and an edge length distribution $\mathbf{p}$ such that $\bar{C}(T \mid T^*, \mathbf{p}) \leq \bar{C}(T^* \mid T^*, \mathbf{p})$.*

We provide a full proof in Appendix A.2, but we give a sketch of the main idea here. An edge $e$ that is shared between the two trees may be more similar to the unique split of a candidate tree $T$ than that of the true tree $T^*$, such that the edge is more likely to generate patterns that are assigned to the unique split of $T$ than $T^*$ at minimum cost. Therefore, for that edge, the expected cost of $T$ is lower than that of $T^*$. The three-leaf example is a minimal case demonstrating this, where the generating edge $e_c = [0\,0\,1]$ was strictly more similar to the unique split $[0\,1\,1]$ of $T$ than $[1\,1\,0]$ of $T^*$. The proof in the Appendix guarantees that at least one such edge $e$ exists for any pair of neighbors $T_1, T_2$ such that $\delta_e(T_1, T_2) \neq 0$.

Lemma 10 provides not just a generic statement of existence, but also the basis for a constructive procedure for generating explicit counterexamples for any pair of neighbor trees: Given *any* neighbor pair $(T_1, T_2)$, an inconsistent edge distribution can be found by (a) computing all $\delta_e$ values by enumerating the $2^n$ observed patterns, and (b) solving for an edge distribution $\mathbf{p}$ such that $\sum_e p_e \delta_e < 0$. Full details of the procedure are given in Appendix B.1, and an implementation that generates random examples of consistency violations under different edge length priors is available at `https://github.com/shahcompbio/phylo_inconsistency`.

## 3.5 The estimator can be positively misleading

Inconsistency, as established above, means only that the true tree fails to be the unique minimizer. A more severe form is when having more data is actively counterproductive, because the estimator converges to a specific incorrect tree with increasing data, known as estimator being *positively misleading* [18].

▶ **Definition 11** (Positively misleading). *The MFCP objective is* positively misleading *if the estimator converges to a unique tree $T \neq T^*$ rather than the true tree $T^*$ as the number of mutations increases.*

▶ **Proposition 12** (The MFCP is positively misleading). *For any weights $w_N, w_P > 0$ and any error probabilities $\alpha, \beta \in (0, 0.5)$, there exist a true tree $T^*$ and an edge length distribution $\mathbf{p}$ for which the MFCP objective is positively misleading.*

The proof of Proposition 12 is given in Appendix A.3. Briefly, it uses the counterexample from Lemma 9, with a constructed $\mathbf{p}$ to ensure that the minimizing tree is unique.

## 3.6 Inconsistency extends beyond nearest neighbors

Our counterexamples so far use NNI neighbors, but the phenomenon is far more general. By Lemma 8, a pair of trees $T_1, T_2$ can escape inconsistency only if $\delta_e(T_1, T_2) = 0$ on *every* shared edge. Each such expectation is a piecewise polynomial in the parameters $(\alpha, \beta, w_P, w_N)$, and the condition $\delta_e(T_1, T_2) = 0$ for all $e$ is an over-determined system of equations (at least $n+1$ equations in three free parameters). Thus, the exact, simultaneous vanishing of the per-edge cost expectations across all shared edges, in general, fails for all but a measure-zero set of parameters $(\alpha, \beta, w_P, w_N)$.

An exception to this is a structurally special family of pairs of trees whose shared-edge cost functionals cancel identically, so that $\delta_e(T_1, T_2) = 0$ for every shared edge $e$. One such example is the balanced quartet $((1,2),(3,4))$ versus $((1,3),(2,4))$. We do not fully

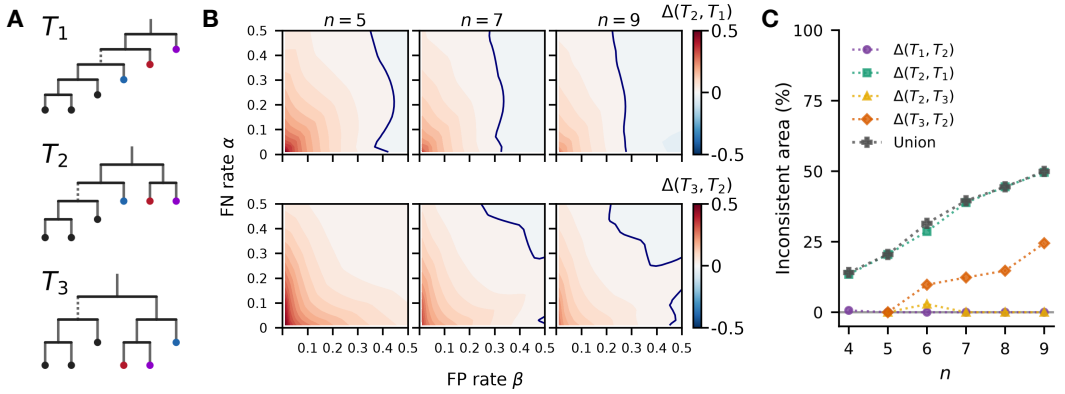



**Figure 4 Consistency violations under equal edge lengths.** (**A**) Three tree topologies on $n$ leaves, named by the number of leaves on the right child of the root: $T_1$, $T_2$ and $T_3$. Colored leaves are shared across all three trees, indicating which leaves change position. Increasing $n$ adds leaves to the left subtree. (**B**) Heatmaps of the expected cost difference over a grid of error probabilities $(\alpha, \beta)$, shown for $n \in \{5, 7, 9\}$ leaves. Top row: $\Delta_{\mathbf{p}}(T_2, T_1)$. Bottom row: $\Delta_{\mathbf{p}}(T_3, T_2)$. The blue line separates positive and negative regions, where the candidate has lower expected cost than the true tree. (**C**) Area of the inconsistent region as a percentage of the $\alpha$-$\beta$ parameter space, across all tested $n$. Each line corresponds to one direction of comparison: $\Delta_{\mathbf{p}}(T_2, T_1) < 0$ (purple), $\Delta_{\mathbf{p}}(T_1, T_2) < 0$ (green), $\Delta_{\mathbf{p}}(T_3, T_2) < 0$ (amber), $\Delta_{\mathbf{p}}(T_2, T_3) < 0$ (orange); black: union across all four directions.

characterize this family, but note that at minimum, the pair of trees must be isomorphic (although this is not a sufficient condition, see $n = 3$ counterexample). Thus the proportion of such pairs is relatively small, especially as $n$ grows.

Consequently, guaranteed consistency is highly constrained and fragile in tree space, and counterexamples are available for nearly all pairs of trees and nearly all parameter values.

# 4 The prevalence of inconsistency under reasonable model parameters

## 4.1 Equal edge lengths do not guarantee consistency

Equal edge lengths is the most favorable candidate for consistency. Under the edge concentration argument, inconsistency requires only that the set $S^- = \{e : \delta_e < 0\}$ is non-empty; arbitrary edge lengths allow mass to be redistributed toward $S^-$ and away from positively-contributing edges, making the negative contributions dominate. Equal edge lengths removes this freedom: the weighted sum $\Delta_{\mathbf{p}} = \sum_e p_e \delta_e$ becomes a simple average, so inconsistency can only occur if the negative-$\delta_e$ edges outweigh the positive ones across the entire tree, a much stricter condition.

For $n \geq 4$, we demonstrate computationally that consistency can still fail for some pairs of trees under equal edge lengths, and that the space of error parameters where this occurs is non-trivial and grows with $n$. We use three trees at each $n$ (Figure 4A): $T_1$ (full caterpillar), $T_2$ (root has a 2-leaf right child), and $T_3$ (root has a 3-leaf right child), where each tree is a neighbor of the previous one. For each tree size $n \in [4, 9]$ (for $T_1/T_2$) and $n \in [5, 9]$ (for $T_2/T_3$), we computed all four pairwise cost differences under equal edge lengths across a 25×25 grid of error probabilities $(\alpha, \beta)$, with weights matched to the error rates. We show that inconsistency occurs in all four directions, but asymmetrically (Figure 4B,C). $T_2$ beats $T_1$ over a substantial and growing region of parameter space, while $T_1$ beats $T_2$ only marginally: at $n = 4$, $T_1$ is preferred over $T_2$ for approximately 1% of parameter values, and

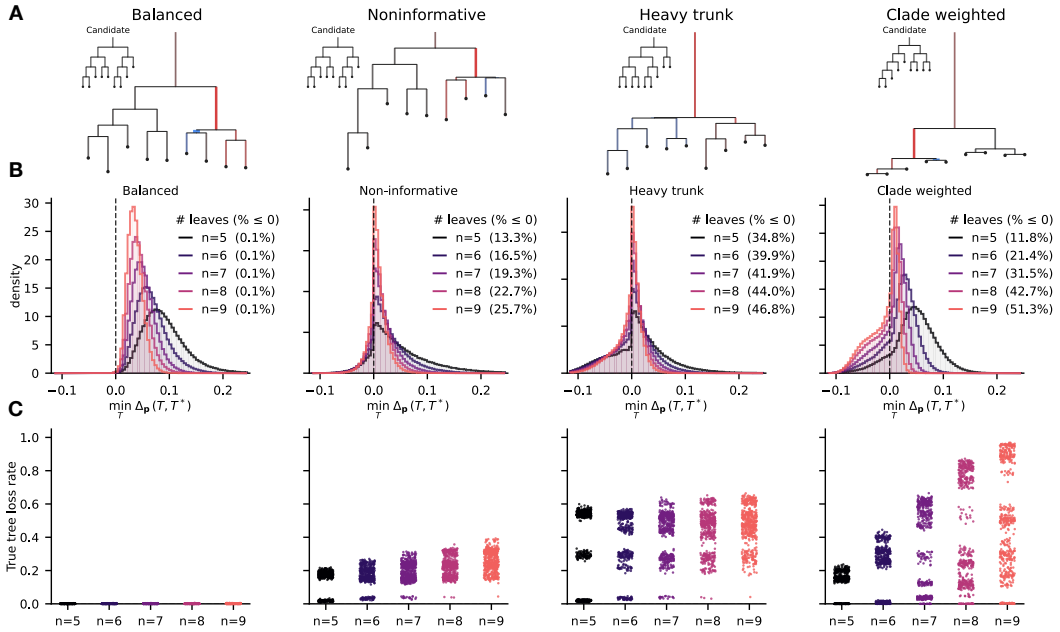



**Figure 5 Prevalence of inconsistency under random edge length draws.** Simulation results for trees with $n$ leaves under four different edge length priors (left to right): balanced, non-informative, heavy trunk, and clade-weighted. (**A**) An example tree topology with $n = 9$ leaves. Each panel shows the same topology under an edge length distribution from each prior, where a candidate tree (shown as an inset) beats the true tree. Edge color indicates the sign of $\delta_e(T, T^*)$ relative to the winning candidate shown in the inset (red: negative, favoring the candidate; blue: positive, favoring the true tree), and edge width indicates magnitude. Edge length is proportional to $p_e$. (**B**) 500 trees are simulated for each $n \in [5, 9]$, and for each tree, 1000 edge length draws are taken under each model. For each tree and edge length draw, we compute the minimum margin $\min_T \Delta_{\mathbf{p}}(T, T^*)$ across all neighbors $T$ of the true tree $T^*$, which is negative if at least one neighbor has a lower cost. For each model, the distribution of minimum margins is shown with the fraction of draws with a negative minimum margin in parentheses. (**C**) The distribution of true tree *loss rates*, i.e., the fraction of draws under which at least one neighbor beats the true tree. Each point corresponds to a single tree and shows the average loss rate across all draws.

this advantage disappears at larger $n$. Similarly, $T_3$ beats $T_2$ over a non-trivial region, while $T_2$ beats $T_3$ only at $n = 6$ for approximately 4% of parameter values. Inconsistency in general is concentrated at high error rates, and the space of error parameters where violations occur grows with $n$ (Figure 4C); at $n = 9$ more than 50% of parameter values produce inconsistency in at least one direction, and the space does not appear to be saturating. This result is still weaker than the theoretical results, which cover *all* error parameters; however, these analyses were limited to a small set of tree topologies and tree sizes. It thus remains open whether any error regime fully restores consistency under equal edge lengths for trees of arbitrary size.

## 4.2 Inconsistency is common under non-pathological edge-length distributions

To evaluate how often such counterexamples occur under biologically motivated edge length distributions, we simulated trees and edge length distributions using a Dirichlet distribution under a set of four different models determining parameters $\vec{\alpha}$ : (a) *Balanced*: balanced edge length distribution, (b) *Non-informative*: uniform on the unit simplex, (c) *Heavy Trunk*: a

long trunk with uniform internal edges, and (d) *Clade-weighted*: $p_e \propto |D_e|^2$ (Figure 5A). Trees were generated with $n \in [5, 9]$ leaves, with the maximum $n$ limited by the computational cost of enumerating all $2^n$ observed patterns for each tree-edge length combination. For each $n$ value, we simulated 500 trees, and for each tree, we took 1000 draws of edge length distributions under each model, for a total of 2,000,000 tree-edge length-model combinations. For each simulated tree and edge length distribution, we computed $\Delta_{\mathbf{p}}(T, T^*)$ for all NNI neighbors $T$ of the true tree $T^*$. NNI neighbors were chosen as a relevant set of candidate trees as we expect them to be most likely to be preferred over the true tree, and the number of NNI neighbors scales linearly with $n$, making the analysis tractable. For further details on the simulation procedures, see Appendix B.2.

For each tree-edge length combination, we computed the minimum margin $\min_T \Delta_{\mathbf{p}}(T, T^*)$ across all neighbors $T$ of the true tree $T^*$, which is negative if at least one neighbor achieves lower expected cost than the true tree under that edge length distribution. As expected, balanced edge lengths perform best: across all $n$, only $\approx 0.1\%$ of draws produced any inconsistent neighbor (Figure 5B). Under the non-informative prior, inconsistency is substantially more common, and increases with $n$: at $n = 5$, 13.3% of draws produced at least one inconsistent neighbor, while at $n = 9$, this fraction increased to 25.7% (Figure 5B). The heavy trunk and clade-weighted priors also show increasing inconsistency with $n$, but at higher rates: 34.8% and 11.8% at $n = 5$ under the heavy trunk and clade-weighted priors respectively, increasing to 46.8% and 51.3% at $n = 9$ (Figure 5B).

The distributions show qualitatively distinct shapes across models (Figure 5B), with the heavy-trunk and clade-weighted models showing a pronounced concentration of mass in the negative range that increases with $n$. Notably, except under the balanced model, even when the true tree is preferred over all neighbors, the margin is often small, suggesting that in finite samples the true tree may frequently be indistinguishable from its neighbors. Moreover, at larger $n$ under the heavy-trunk and clade-weighted models, the negative tail of the distribution extends further than the positive tail, indicating that when a neighbor beats the true tree, it often does so by a larger margin than the true tree achieves when it wins.

## 4.3 Susceptibility to inconsistency is dependent on tree topology and edge length distribution

We next examined the heterogeneity across topologies in their susceptibility to inconsistency. For each tree with $n = 9$, we computed the *true tree loss rate*: the fraction of the 1000 edge length draws under which at least one neighbor beats the true tree (Figure 5C and Figure A1A–C). Loss rates grow with the informativeness of the prior: nearly all trees stay below 0.5% under balanced edge lengths but reach medians of 0.248, 0.464, and 0.503 under the non-informative, heavy-trunk, and clade-weighted priors respectively, with the clade-weighted prior producing the widest spread (from near 0 to near 1). Under every structured prior essentially no tree is immune, and even under balanced edge lengths only 40% of trees never lost. The spread of loss rates widens with $n$, as some topologies become more susceptible and others more robust as tree space grows.

Strikingly, the topologies most susceptible under one prior are largely uncorrelated with those most susceptible under another (Figure A1A–C): trees form well-separated clusters in pairwise loss-rate space that subdivide further at larger $n$. Susceptibility is therefore not a fixed property of topology but is jointly determined by the tree and the edge-length distribution, with different priors placing the burden of inconsistency on different regions of tree space. One exception is the caterpillar topology, which consistently achieves among the lowest loss rates across all priors and tree sizes (Section 4.4).

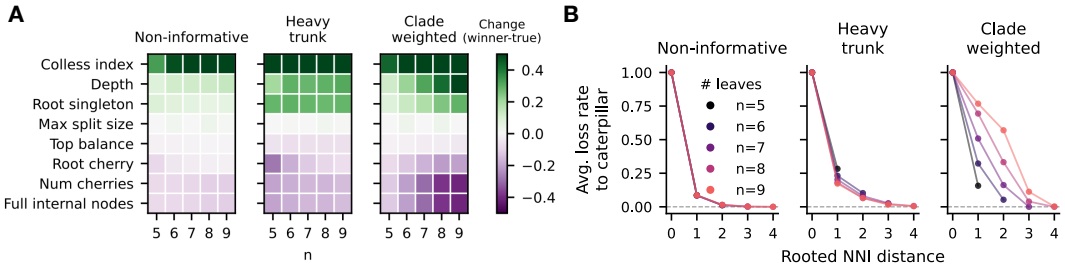



**Figure 6 Topological bias toward caterpillar-like trees at realistic scDNA error rates.** Simulations use error rates $\alpha = 0.2$, $\beta = 0.01$ and three edge-length priors (Non-informative, Heavy trunk, Clade-weighted); the Balanced prior is omitted here as inconsistency is negligible in that regime. (**A**) Mean difference in topological features between winning neighbors and the true tree, averaged over all edge-length draws, across tree sizes $n \in \{5, \ldots, 9\}$ and the three priors. Positive values (red) indicate the winning neighbor tends to have a higher value of that feature than the true tree; negative (blue) indicates the reverse. (**B**) Rate at which a heuristic closest caterpillar beats the true tree in expected flip cost, as a function of the NNI distance between the caterpillar and the true tree. Uses the same error rates and edge-length priors as Panel A, but instead of comparing each true tree against all of its neighbors, each true tree is compared against a single fixed candidate: the heuristic closest caterpillar (see Appendix B.2.1), which may be more than one NNI move away. Each point is the fraction of edge-length draws where the caterpillar wins, averaged over all true trees at a given NNI distance, stratified by tree size and edge-length prior.

The clustering suggests that loss rates are driven by discrete topological features that differ across priors. We fit random forest models predicting per-tree loss rate from eight topological features under each prior, excluding the balanced model, whose loss rates showed no meaningful variation across trees (see Appendix B.2.1 for details). Predictive accuracy was high in all three cases ($R^2 = 0.93$, 0.97, and 0.98 for the non-informative, heavy-trunk, and clade-weighted priors; Figure A1D), confirming that susceptibility is systematically related to tree shape. The predictive features differed across priors in interpretable ways: global balance measures (e.g., number of cherries, Colless index) under the non-informative prior; root-proximal features (root split balance, whether the root has a singleton child) under the heavy-trunk prior, reflecting the influence of the trunk edge; and a mix of balance and depth features with non-monotonic effects under the clade-weighted prior, likely reflecting the dependence of its edge lengths on clade sizes. While the relationship between topology and susceptibility is complex and prior-dependent, identifiable structural properties of the tree thus interact with the edge-length distribution to determine which topologies are most at risk.

## 4.4 Bias toward caterpillar-like trees at realistic scDNA error rates

We examined which topological features systematically differ between neighbors that beat the true tree and the true tree itself (Figure 6A). These simulations use error rates calibrated to single-cell sequencing data ($\alpha = 0.2$, $\beta = 0.01$), where false negatives are substantially more common than false positives. In this regime, winning neighbors consistently had higher cherry counts and lower Colless indices than the true tree across all tested tree sizes and edge-length priors, indicating that the favored topologies tend to be more caterpillar-like. This is consistent with the observation that the caterpillar topology has the lowest loss rate across all priors and tree sizes in this regime (Figure 5C).

We note that this directional bias is specific to this error-rate regime and should not be taken as a general property of the inconsistency phenomenon. Under more symmetric or reversed error rates the preferred topologies shift. This is highly evident in the equal edge-lengths analysis (Section 4.1): at high false positive rates, the bias strongly favors the more balanced trees. Thus, the parameter dependence is not a fixed artifact of the objective but changes with the error profile of the data.

### 4.5 Inconsistency extends beyond neighboring tree topologies

In Section 3.6, we argued that inconsistency is not confined to NNI neighbors of the true tree, but can extend to more distant regions of tree space. We now provide empirical evidence that this is the case, by comparing the true tree against a candidate that is more topologically distant than its immediate neighbors. Because the caterpillar consistently achieves the lowest loss rate in this error regime, we used a heuristic closest caterpillar (see Appendix B.2.1 for construction details) as a fixed candidate and computed how often it beat the true tree as a function of NNI distance. Even when the candidate is several splits away from the true tree, it achieves lower expected cost in a non-trivial fraction of cases (Figure 6B). This is most pronounced under the clade-weighted prior, where the caterpillar beats the true tree in $\approx 50\%$ of cases at an NNI distance of 2 and $\approx 15\%$ at an NNI distance of 3 when $n = 9$, and the proportions increase with $n$. These results are a lower bound on the severity of long-range inconsistency: the caterpillar labeling used is a heuristic, and other labelings or topologically distant candidates may beat the true tree even more frequently, but were not evaluated due to computational constraints. Nonetheless, this suggests that inconsistency is not confined to small topological perturbations of the true tree but can extend to more distant regions of tree space.

## 5 Discussion

We studied the consistency of perfect phylogeny error-correction objectives under observation noise. Our central result is a general inconsistency phenomenon: hard-assignment objectives such as the minimum-cost perfect phylogeny (MFCP) objective and its variants are not guaranteed to uniquely favor the true evolutionary tree in expectation and the failure can be positively misleading: increasing the number of observed mutations can strengthen the support for an incorrect tree. This failure arises from the interaction between observation noise and the structure of the optimization objective itself. It can arise even under equal edge lengths, balanced trees, and symmetric error rates, and is therefore not a consequence of pathologically extreme parameter choices or topologies.

A key feature of the inconsistency phenomenon is that violations are systematic rather than random. Our computational results show that some topologies may be systematically favored across a wide range of edge length distributions. The pattern of which topologies are favored correlates with identifiable structural features, but we do not yet have a complete characterization of these features or how they interact with the observation rates $(\alpha, \beta)$ and edge-length distribution to produce bias. While we have shown that trees can be beaten by candidates that are several splits away, one important open question is a more complete characterization of how far the global optimum typically is from the true tree in practice.

If systematic biases operate across an entire cohort of samples, they may be misinterpreted as recurrent biological signal when they are in fact artifacts of the objective. The direction of bias depends on the observation error regime: under high false negative rates, as is common in single-cell sequencing, the bias favors more linear topologies; under high false positive rates,

balanced topologies tend to be preferred instead. The distinction between branching and linear tumor phylogenies has been a topic of considerable interest in cancer genomics [12, 13], and some methods have used error-correction objectives specifically to ask whether a linear phylogeny is compatible with observed data [3, 49]. Understanding when and how biases in the objective operate is directly relevant for interpreting the results of such analyses.

We wish to emphasize that this paper is not a call to abandon these error-correction objectives in practice. Inconsistent estimators are widely used and often adequate for their intended purpose. While our results show that inconsistency can arise under biologically motivated edge lengths and error rates calibrated to scDNA sequencing, the extent to which it affects biological interpretation in real datasets remains an important direction for future work.

As a consistent alternative, we speculate that maximum-likelihood approaches that marginalize over mutation assignments and edge lengths may be less susceptible to this phenomenon, as they do not rely on a single hard assignment of observed patterns to tree splits. However, marginal approaches often carry additional costs: naive implementations of marginal likelihoods are computationally expensive as they require exhaustive enumeration of all values. Developing consistent and computationally practical alternatives to hard-assignment objectives, and characterizing the conditions under which they are needed, remain important open problems.

## References


1 Richa Agarwala and David Fernández-Baca. A polynomial-time algorithm for the perfect phylogeny problem when the number of character states is fixed. In *Proceedings of 1993 IEEE 34th Annual Foundations of Computer Science*, pages 140–147. IEEE, 1993. doi:10.1109/SFCS.1993.366873.

2 Pranjal Awasthi, Avrim Blum, Jamie Morgenstern, and Or Sheffet. Additive approximation for near-perfect phylogeny construction. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 25–36. Springer, 2012. doi:10.1007/978-3-642-32512-0_3.

3 Erfan Sadeqi Azer, Mohammad Haghir Ebrahimabadi, Salem Malikić, Roni Khardon, and S Cenk Sahinalp. Tumor phylogeny topology inference via deep learning. *Iscience*, 23(11), 2020.

4 Niko Beerenwinkel, Roland F Schwarz, Moritz Gerstung, and Florian Markowetz. Cancer evolution: mathematical models and computational inference. *Systematic biology*, 64(1):e1–e25, 2015.

5 Sebastian Böcker, Quang Bao Anh Bui, François Nicolas, and Anke Truss. Intractability of the minimum-flip supertree problem and its variants. *arXiv preprint*, 2011. arXiv:1112.4536.

6 Sebastian Böcker, Quang Bao Anh Bui, and Anke Truss. An improved fixed-parameter algorithm for minimum-flip consensus trees. In *International Workshop on Parameterized and Exact Computation*, pages 43–54. Springer, 2008.

7 John D Bridgers, Jan Hoinka, S Cenk Sahinalp, Salem Malikic, Teresa M Przytycka, and Funda Ergun. Improved algorithms for bi-partition function computation. In *25th International Conference on Algorithms for Bioinformatics (WABI 2025)*, pages 5:1–5:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.WABI.2025.5.

8 Malte Brinkmeyer, Thasso Griebel, and Sebastian Böcker. Flipcut supertrees: towards matrix representation accuracy in polynomial time. *Algorithmica*, 67(2):142–160, 2013. doi:10.1007/S00453-012-9698-3.

9 Duhong Chen, Oliver Eulenstein, David Fernandez-Baca, and Michael Sanderson. Minimum-flip supertrees: complexity and algorithms. *IEEE/ACM transactions on computational biology and bioinformatics*, 3(2):165–173, 2006. doi:10.1109/TCBB.2006.26.

**10** Markus Chimani, Sven Rahmann, and Sebastian Böcker. Exact ilp solutions for phylogenetic minimum flip problems. In *Proceedings of the First ACM International Conference on Bioinformatics and Computational Biology*, pages 147–153, 2010. doi:10.1145/1854776.1854800.

**11** Simone Ciccolella, Camir Ricketts, Mauricio Soto Gomez, Murray Patterson, Dana Silverbush, Paola Bonizzoni, Iman Hajirasouliha, and Gianluca Della Vedova. Inferring cancer progression from single-cell sequencing while allowing mutation losses. *Bioinformatics*, 37(3):326–333, 2021. doi:10.1093/BIOINFORMATICS/BTAA722.

**12** Ibiayi Dagogo-Jack and Alice T Shaw. Tumour heterogeneity and resistance to cancer therapies. *Nature reviews Clinical oncology*, 15(2):81–94, 2018.

**13** Alexander Davis, Ruli Gao, and Nicholas Navin. Tumor evolution: Linear, branching, neutral or punctuated? *Biochimica et Biophysica Acta (BBA)-Reviews on Cancer*, 1867(2):151–161, 2017.

**14** James H Degnan, Michael DeGiorgio, David Bryant, and Noah A Rosenberg. Properties of consensus methods for inferring species trees from gene trees. *Systematic Biology*, 58(1):35–54, 2009.

**15** James H Degnan and Noah A Rosenberg. Discordance of species trees with their most likely gene trees. *PLoS genetics*, 2(5):e68, 2006.

**16** Mohammadamin Edrisi, Hamim Zafar, and Luay Nakhleh. A combinatorial approach for single-cell variant detection via phylogenetic inference. *bioRxiv*, page 693960, 2019.

**17** Mohammed El-Kebir. Sphyr: tumor phylogeny estimation from single-cell sequencing data under loss and error. *Bioinformatics*, 34(17):i671–i679, 2018. doi:10.1093/BIOINFORMATICS/BTY589.

**18** Joseph Felsenstein. Cases in which parsimony or compatibility methods will be positively misleading. *Systematic zoology*, pages 401–410, 1978.

**19** Joseph Felsenstein. Inferring phylogenies. In *Inferring phylogenies*, pages 664–664. Sinauer Associates, 2004.

**20** Daniel W Feng and Mohammed El-Kebir. Dolphyin: A combinatorial algorithm for identifying 1-dollo phylogenies in cancer. In *25th International Conference on Algorithms for Bioinformatics (WABI 2025)*, pages 9:1–9:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.WABI.2025.9.

**21** Ronald A Fisher. On the mathematical foundations of theoretical statistics. *Philosophical transactions of the Royal Society of London. Series A, containing papers of a mathematical or physical character*, 222(594-604):309–368, 1922.

**22** Walter M Fitch. Toward defining the course of evolution: minimum change for a specific tree topology. *Systematic Biology*, 20(4):406–416, 1971.

**23** Dan Gusfield. Efficient algorithms for inferring evolutionary trees. *Networks*, 21(1):19–28, 1991. doi:10.1002/NET.3230210104.

**24** Michael D Hendy and David Penny. A framework for the quantitative study of evolutionary trees. *Systematic zoology*, 38(4):297–309, 1989.

**25** Akhil Jakatdar, Palash Sashittal, and Benjamin J Raphael. Deriving longitudinal tumor phylogenies from single-cell sequencing data. *bioRxiv*, pages 2025–08, 2025.

**26** Junhyong Kim. General inconsistency conditions for maximum parsimony: effects of branch lengths and increasing numbers of taxa. *Systematic Biology*, 45(3):363–374, 1996.

**27** Motoo Kimura. The number of heterozygous nucleotide sites maintained in a finite population due to steady flux of mutations. *Genetics*, 61(4):893, 1969.

**28** Laura Salter Kubatko and James H Degnan. Inconsistency of phylogenetic estimates from concatenated data under coalescence. *Systematic biology*, 56(1):17–24, 2007.

**29** Liang Liu and Scott V Edwards. Phylogenetic analysis in the anomaly zone. *Systematic biology*, 58(4):452–460, 2009.

**30** Adiesha Liyanage, Robyn Burger, Allison Shi, Braeden Sopp, Binhai Zhu, and Brendan Mumey. Essentcell: Discovering essential evolutionary relations in noisy single-cell data. *bioRxiv*, pages 2025–04, 2025.

31 Jian Ma, Aakrosh Ratan, Brian J Raney, Bernard B Suh, Webb Miller, and David Haussler. The infinite sites model of genome evolution. *Proceedings of the National Academy of Sciences*, 105(38):14254–14261, 2008. doi:10.1073/PNAS.0805217105.

32 Salem Malikic, Katharina Jahn, Jack Kuipers, S Cenk Sahinalp, and Niko Beerenwinkel. Integrative inference of subclonal tumour evolution from single-cell and bulk sequencing data. *Nature communications*, 10(1):2750, 2019.

33 Salem Malikic, Farid Rashidi Mehrabadi, Simone Ciccolella, Md Khaledur Rahman, Camir Ricketts, Ehsan Haghshenas, Daniel Seidman, Faraz Hach, Iman Hajirasouliha, and S Cenk Sahinalp. Phiscs: a combinatorial approach for subperfect tumor phylogeny reconstruction via integrative use of single-cell and bulk sequencing data. *Genome research*, 29(11):1860–1877, 2019.

34 Fábio K Mendes and Matthew W Hahn. Why concatenation fails near the anomaly zone. *Systematic biology*, 67(1):158–169, 2018.

35 Sebastien Roch, Michael Nute, and Tandy Warnow. Long-branch attraction in species tree estimation: inconsistency of partitioned likelihood and topology-based summary methods. *Systematic biology*, 68(2):281–297, 2019.

36 Sebastien Roch and Mike Steel. Likelihood-based tree reconstruction on a concatenation of aligned sequence data sets can be statistically inconsistent. *Theoretical population biology*, 100:56–62, 2015.

37 Edith M Ross and Florian Markowetz. Onconem: inferring tumor evolution from single-cell sequencing data. *Genome biology*, 17(1):69, 2016.

38 Erfan Sadeqi Azer, Farid Rashidi Mehrabadi, Salem Malikić, Xuan Cindy Li, Osnat Bartok, Kevin Litchfield, Ronen Levy, Yardena Samuels, Alejandro A Schäffer, E Michael Gertz, et al. Phiscs-bnb: a fast branch and bound algorithm for the perfect tumor phylogeny reconstruction problem. *Bioinformatics*, 36(Supplement_1):i169–i176, 2020. doi:10.1093/BIOINFORMATICS/BTAA464.

39 Sohrab Salehi, Fatemeh Dorri, Kevin Chern, Farhia Kabeer, Nicole Rusk, Tyler Funnell, Marc J Williams, Daniel Lai, Mirela Andronescu, Kieran R Campbell, et al. Cancer phylogenetic tree inference at scale from 1000s of single cell genomes. *Peer Community Journal*, 3, 2023.

40 Gryte Satas, Matthew A. Myers, and Sohrab P. Shah. shahcompbio/phylo_consistency. Software, swhId: swh:1:dir:4b04560c2d51f196ba4119f87d0f4f858d3783e6 (visited on 2026-08-12). URL: https://github.com/shahcompbio/phylo_inconsistency, doi:10.4230/artifacts.27634.

41 Gryte Satas, Simone Zaccaria, Geoffrey Mon, and Benjamin J Raphael. Scarlet: single-cell tumor phylogeny inference with copy-number constrained mutation losses. *Cell systems*, 10(4):323–332, 2020.

42 Susanne Schulmeister. Inconsistency of maximum parsimony revisited. *Systematic biology*, 53(4):521–528, 2004.

43 Michael A Steel, Michael D Hendy, and David Penny. Parsimony can be consistent! *Systematic biology*, 42(4):581–587, 1993.

44 David L Swofford, Peter J Waddell, John P Huelsenbeck, Peter G Foster, Paul O Lewis, and James S Rogers. Bias in phylogenetic estimation and its relevance to the choice between parsimony and likelihood methods. *Systematic biology*, 50(4):525–539, 2001.

45 Naoko Takezaki and Masatoshi Nei. Inconsistency of the maximum parsimony method when the rate of nucleotide substitution is constant. *Journal of molecular evolution*, 39(2):210–218, 1994.

46 Cuong V Than and Noah A Rosenberg. Consistency properties of species tree inference by minimizing deep coalescences. *Journal of Computational Biology*, 18(1):1–15, 2011. doi:10.1089/CMB.2010.0102.

47 Aad W Van der Vaart. *Asymptotic statistics*, volume 3. Cambridge university press, 2000.

48 Yong Wang and Nicholas E Navin. Advances and applications of single-cell sequencing technologies. *Molecular cell*, 58(4):598–609, 2015.

49 Leah L Weber and Mohammed El-Kebir. Distinguishing linear and branched evolution given single-cell dna sequencing data of tumors. *Algorithms for Molecular Biology*, 16(1):14, 2021. doi:10.1186/S13015-021-00194-5.

50 Yi-Chieh Wu, Matthew D Rasmussen, Mukul S Bansal, and Manolis Kellis. Most parsimonious reconciliation in the presence of gene duplication, loss, and deep coalescence using labeled coalescent trees. *Genome research*, 24(3):475, 2014.

51 Yufeng Wu. Accurate and efficient cell lineage tree inference from noisy single cell data: the maximum likelihood perfect phylogeny approach. *Bioinformatics*, 36(3):742–750, 2020. doi:10.1093/BIOINFORMATICS/BTZ676.

52 Qing Yang, Yuheng Liu, Jijian Yang, Xing Wu, Zhirui Yang, Yonghe Xia, Yunchao Zheng, Jinhong Lu, Mengdie Yao, Yiheng Du, et al. Phylosolid: Robust phylogeny reconstruction from single-cell data despite inherent error and sparsity. *bioRxiv*, pages 2026–02, 2026.

53 Hamim Zafar, Anthony Tzen, Nicholas Navin, Ken Chen, and Luay Nakhleh. Sifit: inferring tumor trees from single-cell sequencing data under finite-sites models. *Genome biology*, 18(1):178, 2017.


# A Proofs and derivations

## A.1 Fisher Consistency and Asymptotic Consistency

Remark 6 asserts that, under our generative model, Fisher consistency (Definition 5) is necessary and sufficient for asymptotic consistency. We justify this here, and more generally identify the tree to which the estimator converges. Consider a true tree $T^*$, edge lengths $\mathbf{p}$, and error rates $\alpha, \beta$. For a sample of $m$ i.i.d. mutations, let $\hat{C}_m(T)$ denote the empirical flip cost of tree $T$ and $\bar{C}(T) = \bar{C}(T \mid T^*, \mathbf{p})$ its expectation. The MFCP estimator selects $\hat{T}_m = \operatorname{argmin}_T \hat{C}_m(T)$.

Suppose the expected cost $\bar{C}$ has a unique minimizer $T^{\min}$ over the tree space, i.e. $\bar{C}(T^{\min}) < \bar{C}(T)$ for all $T \neq T^{\min}$. Then $P(\hat{T}_m = T^{\min}) \to 1$ as $m \to \infty$. This follows from Theorem 5.7 of [47], whose three conditions all hold here: (i) the mutations are i.i.d. by assumption; (ii) each per-mutation cost $C(T, \hat{\mathbf{x}})$ is bounded (at most $n$ flips), so in particular has finite expectation; (iii) the tree space is finite, so $\bar{C}$ has a well-separated unique minimizer at $T^{\min}$ (with gap $\gamma = \min_{T \neq T^{\min}}[\bar{C}(T) - \bar{C}(T^{\min})] > 0$).

When Fisher consistency holds, $T^{\min} = T^*$, so $\hat{T}_m \to T^*$ and the estimator is asymptotically consistent. Conversely, suppose Fisher consistency fails, so there exists some $T \neq T^*$ with $\bar{C}(T) \leq \bar{C}(T^*)$. By the law of large numbers $\hat{C}_m(T) \to \bar{C}(T)$ and $\hat{C}_m(T^*) \to \bar{C}(T^*)$, so $\hat{C}_m(T) \leq \hat{C}_m(T^*)$ with probability approaching 1, and the estimator does not converge to $T^*$.

## A.2 Proof of Lemma 10

We use the setup and notation of Lemma 10 throughout this section: $T_1$ and $T_2$ are NNI neighbors with subtrees $t_A, t_B, t_C, t_D$ where $t_A$ and $t_B$ are the exchanged subtrees. $T_1$ has unique split $s_1$, which is ancestral to subtrees $t_A$ and $t_C$, while $T_2$ has unique split $s_2$ which is ancestral to subtrees $t_B$ and $t_C$. Let $A$ be a leaf of subtree $t_A$, and $B$ be a leaf of subtree $t_B$. Without loss of generality, we order mutation patterns $\mathbf{x} = [x_A\ x_B\ \mathbf{z}]$ where $x_A$ and $x_B$ are the state of $A$ and $B$ respectively and $\hat{\mathbf{z}} \in \{0,1\}^{(n-2)}$ is the state of the remaining leaves.

Let $e_A$ and $e_B$ be the edges corresponding to leaves $A$ and $B$. Below we will demonstrate the following:

1. Patterns of the form $\hat{\mathbf{x}} = [1\,0\,\hat{\mathbf{z}}]$ relatively benefit $T_1$ when compared to patterns of the form $\hat{\mathbf{x}} = [0\,1\,\hat{\mathbf{z}}]$.
2. Patterns of the form $\hat{\mathbf{x}} = [1\,0\,\hat{\mathbf{z}}]$ are more likely to be generated by $e_A$ than patterns of the form $\hat{\mathbf{x}} = [0\,1\,\hat{\mathbf{z}}]$.

By symmetry, the opposite of these properties hold for $T_2$ and $e_B$. As a result, $e_A$ is more likely to generate patterns that benefit $T_1$ while $e_B$ is more likely to generate patterns that benefit $T_2$, and consequently $\delta_{e_A}(T_1, T_2) < \delta_{e_B}(T_1, T_2)$.

### Cost Asymmetry

Here we show that for all $\hat{\mathbf{z}}$, the net advantage $C(T_1, \hat{\mathbf{x}}) - C(T_2, \hat{\mathbf{x}})$ strictly decreases between observed pattern $\hat{\mathbf{x}} = [1\,0\,\hat{\mathbf{z}}]$ and $\hat{\mathbf{x}} = [0\,1\,\hat{\mathbf{z}}]$. That is patterns of the form $\hat{\mathbf{x}} = [1\,0\,\hat{\mathbf{z}}]$ relatively benefit $T_1$, compared to patterns of the form $\hat{\mathbf{x}} = [0\,1\,\hat{\mathbf{z}}]$ which relatively benefit $T_2$.

Tree $T_1$ can only achieve a lower cost on a pattern $\hat{\mathbf{x}}$ than $T_2$ if the minimal cost assignment for $\hat{\mathbf{x}}$ is the unique split $s_1$, because $T_2$ shares all other splits in $T_1$. Let $n_A$ be the number of leaves in $t_A$, and let $k_A$ be the number of leaves in $t_A$ with state 1, for some $\hat{\mathbf{x}}$.

$s_1$ can only be the minimum cost assignment if $w_N \cdot k_A \geq w_P \cdot (n_A - k_A)$; that is, the cost of flipping all $1 \to 0$ exceeds the cost of flipping all $0 \to 1$. Otherwise, the trunk of subtree $t_C$ would have lower cost than $s_1$. Note that this is a necessary but not sufficient condition for $s_1$ to be the optimal assignment. Consider patterns of the form $[1\,0\,\hat{\mathbf{z}}]$ and $[0\,1\,\hat{\mathbf{z}}]$ for $\hat{\mathbf{z}} \in \{0,1\}^{n-2}$. Switching from $[1\,0\,\hat{\mathbf{z}}]$ to $[0\,1\,\hat{\mathbf{z}}]$ strictly decreases the number of leaves in $t_A$ with value 1. Therefore, it can never make $s_1$ optimal if it were not already optimal. A symmetric argument holds for $T_2$ and flipping from $[0\,1\,\hat{\mathbf{z}}]$ to $[1\,0\,\hat{\mathbf{z}}]$. As a result, we have that

$$C(T_1, [1\,0\,\hat{\mathbf{z}}]) - C(T_1, [0\,1\,\hat{\mathbf{z}}]) \leq C(T_2, [1\,0\,\hat{\mathbf{z}}]) - C(T_2, [0\,1\,\hat{\mathbf{z}}]). \tag{2}$$

There exists at least one $\hat{\mathbf{z}}$ for which this inequality is strict. Take $\hat{\mathbf{z}}$ with all leaves of $t_B$ and $t_D$ in state 0 and all leaves of $t_C$ in state 1, and choose the number of state-1 leaves in $t_A$ so that the unique split $s_1$ is the unique minimum-cost assignment for $[1\,0\,\hat{\mathbf{z}}]$ but not for $[0\,1\,\hat{\mathbf{z}}]$. Such a choice exists: passing from $[1\,0\,\hat{\mathbf{z}}]$ to $[0\,1\,\hat{\mathbf{z}}]$ flips $A$ from 1 to 0, removing one leaf from $t_A$ and hence from the clade of $s_1$, and there is a number of state-1 leaves in $t_A$ at which this single flip moves $s_1$ from uniquely optimal to non-optimal (tied or beaten by the $t_C$ split). For this $\hat{\mathbf{z}}$, $T_1$ attains via $s_1$ a cost on $[1\,0\,\hat{\mathbf{z}}]$ that $T_2$ cannot match, whereas on $[0\,1\,\hat{\mathbf{z}}]$ any cost $T_1$ attains is also attained by $T_2$ through a shared split. Therefore, $C(T_1, [1\,0\,\hat{\mathbf{z}}]) - C(T_1, [0\,1\,\hat{\mathbf{z}}]) < C(T_2, [1\,0\,\hat{\mathbf{z}}]) - C(T_2, [0\,1\,\hat{\mathbf{z}}])$.

### Probability asymmetry

Under the generating edge $\phi^* = e_A$, leaf $A$ has true state 1 and leaf $B$ has true state 0; under $\phi^* = e_B$ the opposite holds. Let $\gamma = (1-\alpha)(1-\beta) - \alpha\beta$ and $P_{\hat{\mathbf{z}}}$ be the probability of observing $\hat{\mathbf{z}}$. By independence of the observation model, the probabilities of each pattern type are:

| Pattern | $\Pr(\hat{\mathbf{x}} \mid \phi^* = e_A)$ | $\Pr(\hat{\mathbf{x}} \mid \phi^* = e_B)$ | Difference |
|---|---|---|---|
| $[1\,0\,\hat{\mathbf{z}}]$ | $(1-\alpha)(1-\beta)\,P_{\hat{\mathbf{z}}}$ | $\alpha\beta\,P_{\hat{\mathbf{z}}}$ | $+\gamma P_{\hat{\mathbf{z}}}$ |
| $[0\,1\,\hat{\mathbf{z}}]$ | $\alpha\beta\,P_{\hat{\mathbf{z}}}$ | $(1-\alpha)(1-\beta)\,P_{\hat{\mathbf{z}}}$ | $-\gamma P_{\hat{\mathbf{z}}}$ |
| $[1\,1\,\hat{\mathbf{z}}]$ | $(1-\alpha)\beta\,P_{\hat{\mathbf{z}}}$ | $(1-\alpha)\beta\,P_{\hat{\mathbf{z}}}$ | 0 |
| $[0\,0\,\hat{\mathbf{z}}]$ | $\alpha(1-\beta)\,P_{\hat{\mathbf{z}}}$ | $\alpha(1-\beta)\,P_{\hat{\mathbf{z}}}$ | 0 |

For all $\alpha, \beta \in (0, 0.5)$, $\gamma > 0$.

## A.3 Proof of Proposition 12

We use the three-leaf counterexample of Lemma 9. On leaf set $\{A, B, C\}$ there are exactly three rooted binary trees,

$$T^* = ((A, B), C), \qquad T_1 = (A, (B, C)), \qquad T_2 = ((A, C), B),$$

which share every split except their internal cherry: [1 1 0], [0 1 1], and [1 0 1] respectively. Note that candidates $T_1$ and $T_2$ are symmetric under the exchange of leaves $A$ and $B$ with respect to the true tree $T^*$, and thus by the argument of Lemma 9, the generating edge $e_C$ with true pattern [0 0 1] favors both $T_1$ and $T_2$ over $T^*$ by the same amount. By the same argument, the generating leaf edges $e_A$ and $e_B$ with true patterns [1 0 0] and [0 1 0] favor $T_2$ and $T_1$ respectively. Consequently, making $p_{e_C}$ sufficiently large guarantees that both $T_1$ and $T_2$ beat $T^*$, while making $p_{e_B} > p_{e_A}$ ensures that $T_1$ beats $T_2$. As a result, by Appendix A.1 the estimator uniquely converges to $T_1 \neq T^*$ as the number of mutations increases.

### A.4 Derivation of Maximum Likelihood Weight Correspondence

We derive the flip-cost weights $w_P$ and $w_N$ under which the MFCP objective is equivalent to maximizing the joint likelihood over trees and mutation assignments. Consider a single mutation with observed pattern $\hat{\mathbf{x}} \in \{0,1\}^n$. For each edge $e$, let $\mathbf{x} \in \{0,1\}^n$ be the mutation pattern induced by $e$. The log-likelihood of assigning the mutation to edge $e$ is

$$\ell(e) = \sum_i \log \Pr(x_i \mid \hat{x}_i).$$

Let $n^+ = |\{i : \hat{\mathbf{x}}_i = 1\}|$ and $n^- = |\{i : \hat{\mathbf{x}}_i = 0\}|$, which depend only on $\hat{\mathbf{x}}$ (not on $e$). The four cases for leaf $i$ are:

$$\Pr(x_i \mid \hat{x}_i) = \begin{cases} 1-\alpha & \text{if } x_i = 1, \hat{x}_i = 1 \text{ (no flip)} \\ \alpha & \text{if } x_i = 1, \hat{x}_i = 0 \text{ (0}\to\text{1 flip)} \\ \beta & \text{if } x_i = 0, \hat{x}_i = 1 \text{ (1}\to\text{0 flip)} \\ 1-\beta & \text{if } x_i = 0, \hat{x}_i = 0 \text{ (no flip)} \end{cases}$$

Let $n_{P,e} = |\{i : x_i = 1,\, \hat{x}_i = 0\}|$ and $n_{N,e} = |\{i : x_i = 0,\, \hat{x}_i = 1\}|$. Collecting terms:

$$\begin{aligned} \ell(e) &= (n^+ - n_{N,e}) \log(1-\alpha) + n_{P,e} \log \alpha + n_{N,e} \log \beta + (n^- - n_{P,e}) \log(1-\beta) \\ &= n^+ \log(1-\alpha) + n^- \log(1-\beta) - n_{N,e}\big[\log(1-\alpha) - \log\beta\big] \\ &\quad - n_{P,e}\big[\log(1-\beta) - \log\alpha\big] \\ &= \text{constant} \;-\; n_{N,e} \log \tfrac{1-\alpha}{\beta} \;-\; n_{P,e} \log \tfrac{1-\beta}{\alpha}. \end{aligned}$$

In other words, the log likelihood gives a penalty of $\log \frac{1-\beta}{\alpha}$ for each $0 \to 1$ flip and a penalty of $\log \frac{1-\alpha}{\beta}$ for each $1 \to 0$ flip, up to an additive constant that does not depend on the edge assignment. Summing over mutations and maximizing over $(T, \Phi)$ jointly therefore corresponds to the MFCP objective with weights $w_P = \log \frac{1-\beta}{\alpha}$ and $w_N = \log \frac{1-\alpha}{\beta}$.

## B Computational Methods

### B.1 Algorithmic Construction of Counterexamples

We describe a procedure for generating explicit counterexamples to consistency for any pair of neighbor trees, based on the proof of Lemma 10 and the edge concentration argument of Lemma 8. An implementation of this procedure is available at `https://github.com/shahcompbio/phylo_inconsistency`.

1. **Sample a random NNI pair.** Given $n \geq 3$, we sample a random rooted binary tree $T^*$ on $n$ leaves. We then sample an internal edge uniformly at random and apply an NNI operation to obtain the candidate tree $T$.

2. **Compute $\boldsymbol{\delta_e}$.** For each edge $e \in E(T^*)$ we compute $\delta_e(T, T^*)$ by explicit enumeration over all $2^n$ observed patterns $\hat{\mathbf{x}} \in \{0,1\}^n$. If all $\delta_e \geq 0$ (i.e. $T^*$ already dominates pointwise), the roles of $T^*$ and $T$ are swapped, as Lemma 10 guarantees at least one orientation will have a negative delta.
3. **Find an inconsistent edge distribution.** We aim to find an edge distribution subject to $\sum_e p_e \delta_e < 0$ that is not pathologically extreme when possible. We look for the distribution $\mathbf{p}$ that is closest (in KL divergence) to a chosen prior $\mathbf{q}$ subject to $\sum_e p_e \delta_e = 0$, then perturb it to make the sum strictly negative. The minimum-KL problem $\min_p \mathrm{KL}(p\|q)$ subject to $\sum_e p_e \delta_e = 0$ and $\sum_e p_e = 1$ has a closed-form solution via Lagrange multipliers:

   $$p_e^* \;\propto\; q_e \, e^{\lambda^* \delta_e},$$

   where $\lambda^*$ is the unique root of $f(\lambda) = \sum_e p_e^*(\lambda)\, \delta_e = 0$. $f(\lambda)$ is continuous and monotone in $\lambda$ so a root is found efficiently via binary search.
   We use three priors $\mathbf{q}$: (a) *Balanced:* all edges are equally weighted, (b) *Heavy trunk:* all edges are equally weighted except the trunk edge, which is given a higher weight, and (c) *Clade weighted:* $q_e \propto s_e^2$, where $s_e = \sum_i (\mathbf{x}_e)_i$ is the clade size of edge $e$. The clade-weighted prior is inspired by coalescent models of evolution, which predict that edges with larger clades tend to be longer.

## B.2 Simulation procedures

We describe the simulation pipeline used in Section 4.

### Tree generation

For each simulation condition we draw $N$ random rooted binary trees on $n$ leaves.

### Edge-length distributions

We treat the edge-length vector $\mathbf{p} = (p_e)_{e \in E(T^*)}$, $\sum_e p_e = 1$, as a random quantity drawn from a prior distribution. We consider four priors.

1. **Balanced.** $\mathbf{p} \sim \mathrm{Dirichlet}(5)$.
2. **Non-informative.** $\mathbf{p} \sim \mathrm{Dirichlet}(\mathbf{1})$.
3. **Heavy-trunk.** $\mathbf{p} \sim \mathrm{Dirichlet}(\boldsymbol{\alpha})$ where $\alpha_e = 5$ for the trunk edge (the root edge) and $\alpha_e = 1$ for all other edges.
4. **Clade-weighted.** $\mathbf{p} \sim \mathrm{Dirichlet}(\boldsymbol{\alpha})$ where $\alpha_e = |\mathcal{D}_e|^2$.

### NNI comparison

For each sampled true tree $T^*$ and edge-length draw $\mathbf{p}$, we test whether the objective prefers $T^*$ over all of its NNI neighbours, i.e., $\Delta_{\mathbf{p}}(T, T^*) < 0$. For each internal edge $e \in E(T^*)$ and each of its two NNI rearrangements, we compute the margin

$$\Delta_{\mathbf{p}}(T, T^*) \;=\; \sum_{e'} p_{e'}\, \delta_{e'}(T, T^*),$$

where $\delta_{e'}(T, T^*)$ is computed by explicit enumeration over all $2^n$ observed patterns $\hat{\mathbf{x}}$.

## B.2.1 True tree loss rate prediction

To investigate the relationship between tree topology and the loss rate of the true tree $T^*$, we computed a set of topological features for each tree, as follows:

- **Depth.** The maximum distance from the root to any leaf.

- **Balance.** The Colless index, which is the sum over all internal nodes of the absolute difference in the number of descendant leaves in the left and right subtrees
- **Top balance.** The balance at the root node, i.e., the absolute difference in the number of descendant leaves in the left and right subtrees of the root.
- **Root has singleton child.** Whether the root has a child that is a leaf.
- **Root has cherry child.** Whether the root has a child that is an internal node with two leaf children (a cherry).
- **Cherry count.** The number of cherries in the tree, where a cherry is an internal node with two leaf children.
- **Full internal node count.** The number of internal nodes with two internal node children.
- **Max split size.** The size of the largest internal split of the tree, measured by the number of descendant leaves on the smaller side of the split.

A random forest regression model was then trained to predict the loss rate of the true tree $T^*$ from these topological features, separately for each edge-length prior. The importance of each feature was evaluated using permutation importance, which measures the increase in mean squared error when the feature is randomly permuted across trees.

### B.2.2 Heuristic Caterpillar Construction

There are many caterpillar trees on $n$ leaves that differ in their leaf ordering along the spine. Some of these may be more likely to be preferred over the true tree than others, and it is unclear which caterpillar tree would be optimal. We used a heuristic construction to select a caterpillar by selecting the tree whose internal clades closely resemble the clades of the true tree. We extract a leaf ordering from the true tree through a depth-first traversal from the root. At each internal node, the deeper child clade is visited first. This construction is heuristic and doesn't guarantee that the resulting caterpillar is the best possible caterpillar for comparison, but it is a principled way to select a single caterpillar that may be more likely to be preferred over the true tree than a random caterpillar, and it is computationally efficient to construct.

## C Additional Figures

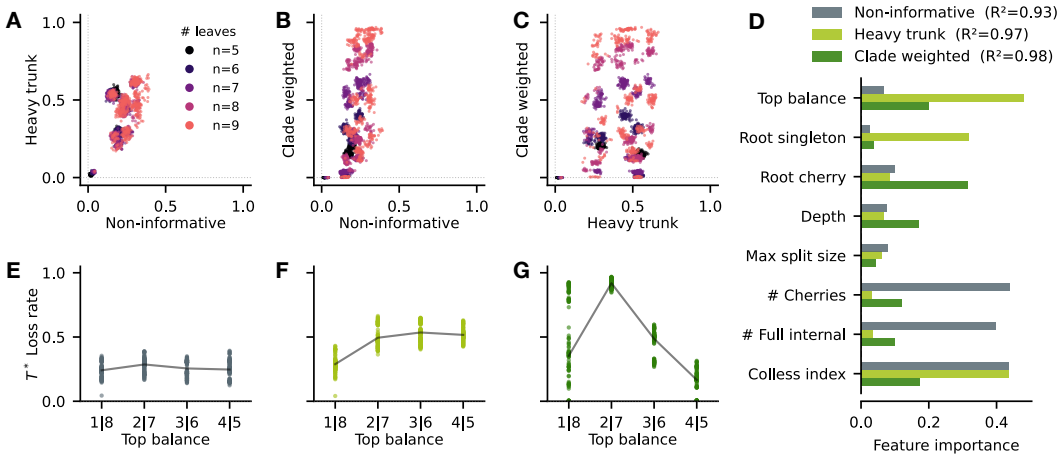



**Figure A1 Prediction of true-tree loss rate from tree topology.** (**A**–**C**) Pairwise scatter plots of true-tree loss rates across all tree topologies, comparing the Non-informative, Heavy trunk, and Clade-weighted edge-length priors. Each point represents one tree topology. (**D**) Permutation feature importance for the topological features used in the random forest models, shown for the three edge-length models with the same color scheme as in panels A–C: Non-informative in grey, Heavy trunk in orange, and Clade-weighted in purple. Cross-validated $R^2$ values for each model are annotated. (**E**–**G**) Distributions of the top-balance feature for the trees under the Non-informative, Heavy trunk, and Clade-weighted models, respectively.

# CoSTAR: Coarse Stem-Topology Alignment of Pseudoknotted RNA Structures by Relation-Constrained Search

**Finn Archinuk**
Department of Biomedical Engineering, University of Alberta, Edmonton, Canada

**Hosna Jabbari**[1]
Department of Biomedical Engineering, University of Alberta, Edmonton, Canada

## Abstract

RNA structural alignment is a central task in comparative RNA analysis, but many efficient methods achieve tractability by restricting the class of admissible structures, often excluding pseudoknots. This exclusion is limiting for viral and regulatory RNAs, where conserved structure can remain informative even when sequence conservation is weak. We introduce a coarse RNA structural alignment algorithm that aligns secondary structures by searching over partial maps between stems rather than nucleotides. Each input structure is decomposed into stems, annotated with nucleotide-level features, and encoded by pairwise topological relations among stems. Alignment is formulated as a cost-minimizing partial stem map with skip operations, and the search tree is pruned by RNA-specific directionality and topological constraints derived from already aligned stems. For the stated cost function and over the class of injective, direction-preserving, topologically consistent stem maps, the search is exact. This shifts the dominant computational dependence from sequence length to the number and arrangement of stems. We evaluated the method on 2100 pairwise alignments sampled from seven Rfam families spanning 40–224 nucleotides and 2–15 stems. Across these benchmarks, the algorithm returned terminal coarse alignments in which every stem was either matched or skipped. We measured running time and search-tree width to characterize performance on diverse family-to-family comparisons. The experiments also show that ordering the input structures affects efficiency: using the structure with more stems as the search-driving structure reduces tree width. The resulting partial stem map is directly interpretable for RNA annotation and can be projected to nucleotide resolution for downstream sequence–structure analysis. The source code for CoSTAR is available at: `https://github.com/TheCOBRALab/CoSTAR`

**2012 ACM Subject Classification** Applied computing → Bioinformatics; Theory of computation → Design and analysis of algorithms

**Keywords and phrases** RNA structural alignment, pseudoknots, RNA secondary structure, branch-and-bound search, stem topology

**Digital Object Identifier** 10.4230/LIPIcs.WABI.2026.24

**Related Version** *Preliminary Version*: `https://doi.org/10.64898/2026.05.22.727263`

**Supplementary Material** *Software (Source Code)*: `https://github.com/TheCOBRALab/CoSTAR` archived at `swh:1:dir:d00d8facb68d10226195689f8d8ff14666d61172`

## 1 Introduction

RNA molecules perform diverse informational, catalytic, and regulatory functions in the cell. Protein-coding mRNAs serve as templates for translation. Non-coding RNAs contribute to translation as ribosomal and transfer RNAs, to RNA processing as spliceosomal and small nucleolar RNAs, to gene regulation as small RNAs and long non-coding RNAs, to

[1] corresponding author


© Finn Archinuk and Hosna Jabbari;
licensed under Creative Commons License CC-BY 4.0



genome organization and defense, and to catalysis as ribozymes [19, 20, 4, 7]. Across these roles, RNA activity often depends not only on primary sequence, but also on structures formed within the molecule and through interactions with other RNAs, DNA, proteins, or small molecules [26, 5, 12]. Thus, the analysis of RNA function often requires methods that compare molecules beyond their primary sequences.

One important level of RNA structural organization is secondary structure, defined by the set of intramolecular base pairs in the molecule [26]. For many structural non-coding RNAs, functional constraints act on the base-pairing pattern at least as strongly as on the primary sequence. Consequently, homologous RNAs may show weak sequence identity while retaining corresponding base pairs. At a conserved base pair, a substitution at one paired position can be accompanied by a substitution at the other position so that base-pair complementarity is maintained. These coordinated changes are compensatory substitutions, and their statistical association across an alignment is referred to as covariation [10]. Comparative RNA methods use such covariation as evidence for conserved secondary structure [8]. Structural alignment is therefore a central computational primitive for comparing ncRNAs, annotating conserved structural elements, and transferring functional or mechanistic hypotheses between related RNAs [3].

The algorithmic difficulty of structural alignment depends strongly on the class of secondary structures handled. If all base pairs are non-crossing, the structure is pseudoknot-free and has a nested topology. This nesting property supports efficient dynamic programming decompositions, tree representations, and grammar-based models [24, 25, 9]. RNAforester, for example, uses a tree representation to compute local similarity between RNA secondary structures [15], whereas BEAR enriches dot-bracket strings with contextual structural symbols and learned substitution scores [18]. However, these efficient approaches are designed for a restricted structural class and do not preserve crossing base-pair dependencies. Although pseudoknots constitute a minority of annotated structures in current datasets (bpRNA estimates that approximately 12% of annotated structures contain pseudoknots [6]), they occur in important viral regulatory mechanisms, including gene expression and replication [2]. Thus, the existing efficient structural comparison methods may miss the biologically important structural motifs that lie outside of their pseudoknot-free class of structures.

This limitation is especially relevant for viral non-coding and regulatory RNA elements. Viral genomes can accumulate substitutions rapidly; as sequence divergence increases, nucleotide-level alignments may fail to identify corresponding stems or structural motifs [21]. At the same time, particular stems and pseudoknotted motifs may be constrained by replication, translation, or regulatory function, and may therefore remain comparable even when the underlying nucleotides differ [2]. Some sequence–structure alignment methods can handle pseudoknotted inputs. LaRA 2, for example, aligns RNA sequences with pseudoknotted secondary structures [28]. Such methods return nucleotide-to-nucleotide alignments. In applications where the primary question is whether two RNAs contain corresponding stems, this resolution is not always necessary. We therefore consider a coarser alignment problem: compute a partial stem map between the stems of two RNA secondary structures, while preserving their relative topological relationships. This stem-level map can be used directly for structural annotation or as a set of anchors for subsequent nucleotide-level sequence–structure alignment.

Motivated by this setting, we formulate RNA structural alignment at the level of stems rather than nucleotides. For each input structure, short bulges and internal loops are first compressed according to a user-defined threshold, and the resulting structure is decomposed into stems. Each stem is represented by a feature vector that includes the number of base

pairs, the number of nucleotides in its two stem arms, and its stacking-energy magnitude. We also precompute a relation array for each structure. For an ordered pair of stems $(A, B)$, this array records the topological relation of $B$ with respect to $A$: 5′-disjoint, 3′-disjoint, inside $A$, containing $A$, crossing $A$ from the 5′ side, or crossing $A$ from the 3′ side. Given two input structures $R^F$ and $R^G$, with ordered stem lists $\mathcal{F}$ and $\mathcal{G}$, a coarse alignment is a partial injective map from stems of $\mathcal{F}$ to stems of $\mathcal{G}$. Stems that are not included in the map are assigned skip costs, and matched stems are assigned costs derived from their feature vectors.

The computational bottleneck is the number of possible partial stem maps. If $\mathcal{F}$ has $f$ stems and $\mathcal{G}$ has $g$ stems, then, without any order constraint, the number of partial injective maps is $\sum_{k=0}^{\min(f,g)} \binom{f}{k}\binom{g}{k}k!$. Indeed, for each $k$, one chooses $k$ stems from $\mathcal{F}$, $k$ stems from $\mathcal{G}$, and an arbitrary bijection between the two chosen sets. CoSTAR restricts the search to direction-preserving maps: if stem $F_i$ is matched before $F_{i'}$, then its image in $\mathcal{G}$ must also occur before the image of $F_{i'}$ in $\mathcal{G}$. Under this restriction, the two chosen $k$-element subsets determine a unique order-preserving bijection. Hence the number of terminal direction-preserving partial maps, before applying topological constraints, is $\sum_{k=0}^{\min(f,g)} \binom{f}{k}\binom{g}{k} = \binom{f+g}{f}$, where the equality follows from Vandermonde's identity. This space remains exponential in the number of stems, and the generated search tree also contains prefixes of such maps. CoSTAR therefore expands partial alignments in the 5′-to-3′ order of $\mathcal{F}$. When assigning the next stem $F_i$, a candidate $G_j$ is considered only if $j$ is larger than every previously matched index in $\mathcal{G}$. The candidate is admissible only if, for every previously matched pair $(F_p, G_q)$, the relations of the next stem $\rho_F(p, i)$ are identical to those of the candidate stems $\rho_G(q, j)$, where $\rho_F(p, i)$ and $\rho_G(q, j)$ denote the corresponding ordered stem relations in the two input structures. These precomputed local consistency checks reduce the branching factor without requiring either structure to be pseudoknot-free.

**Contributions.** We introduce CoSTAR, a relation-constrained search algorithm for coarse stem-topology alignment of RNA secondary structures that may contain pseudoknots. CoSTAR represents each input as an ordered list of stems with nucleotide-derived features and precomputed pairwise topological relations. It then solves a minimum-cost alignment problem over partial injective stem maps with explicit skip operations, using 5′-to-3′ order and relation-array consistency to prune infeasible extensions. We evaluate CoSTAR on 2100 pairwise alignments from seven Rfam families with 2–15 stems, characterizing running time and search-tree width and showing that placing the structure with more stems in the search-driving role empirically reduces search-tree width. The output is an interpretable partial stem map for annotation or downstream nucleotide-level sequence–structure alignment.

## 2 Related Work

RNA comparison methods differ in the information they use, the structural classes they support, and the resolution of the output. Classical sequence–structure alignment methods extend sequence alignment by incorporating base-pairing information into the objective. The Sankoff formulation gives a general framework for simultaneous alignment and folding, but its full dynamic program is computationally expensive [24]. LocARNA implements an efficient Sankoff-style approach for local sequence–structure alignment of pseudoknot-free RNAs and has been used for structure-based clustering of non-coding RNAs [27]. LaRA and LaRA 2 address fixed sequence–structure alignment using combinatorial optimization. In particular, LaRA models sequence–structure alignment as an integer linear program, and LaRA 2

accelerates pairwise alignment by using parallel execution and vectorized alignment kernels [1, 28]. These methods are appropriate when the desired output is a nucleotide-to-nucleotide alignment. Our objective is different: given two fixed structures, we compute a partial stem map between stems. This coarser output records which stems correspond and which are skipped, and can later be projected to nucleotide resolution if needed.

Several methods compare RNA secondary structures primarily by transforming or enriching structural representations. RNAforester represents pseudoknot-free secondary structures as ordered forests and computes local or global structural alignments [15]. BEAR replaces standard dot-bracket notation with a context-aware alphabet that distinguishes paired and unpaired positions according to their structural context, and then compares the resulting strings using learned substitution scores [18]. More recently, bpRNA-align uses feature-specific structural arrays, context-dependent affine gap penalties, and a structural substitution matrix to compute global similarity scores between RNA secondary structures [16]. These approaches provide useful structural similarity measures, but their alignments are defined over nucleotide positions or position-level structural symbols. In contrast, our method searches directly over partial maps between stems, which are the objects returned by the output alignment.

Pseudoknotted structures require representations that do not rely on purely nested topology. ASPRAlign compares secondary structures with arbitrary pseudoknots by transforming them into algebraic or structural RNA trees and computing the ASPRA distance [22]. This provides an efficient structure-only distance for pseudoknotted RNAs. However, the ASPRA distance is a global dissimilarity measure rather than an explicit feature-weighted correspondence between stems. Our method instead keeps pseudoknotted relations as pairwise topological constraints during search, while retaining stem-level quantitative features such as stem length and stacking energy.

Descriptor-based RNA motif search provides another relevant abstraction. In RNArobo, a user specifies a motif descriptor containing sequence and structural constraints, and the algorithm searches genomic sequences for matching regions [23]. Although descriptor search is not a pairwise alignment problem, it illustrates the utility of representing RNA structure in terms of ordered helices and other structural descriptors rather than individual nucleotides. Our reduced topology graph follows this element-level view, but uses it to align two complete structures rather than to search a sequence database for occurrences of a query descriptor.

Topology-based abstractions are also closely related. RNA abstract shapes collapse secondary structures into branching forms, enabling clustering of structures that differ in local base-pair details but share a global organization [13]. RNA-as-Graphs represents RNA secondary structures as graph topologies, including pseudoknotted structures through dual graphs [11]. Such abstractions are intentionally non-injective: distinct structures can map to the same abstract object when they differ in stem length, nucleotide composition, or energetic stability but not in topology. Super N-motifs takes a different alignment-free approach by decomposing structures into simple motifs and constructing vector representations from informative motif combinations [14]. These representations are useful for clustering or large-scale comparison, but they do not directly return a one-to-one correspondence between stems in two RNAs.

Our method occupies an intermediate point between nucleotide-level alignment and alignment-free topology comparison. It supports pseudoknotted inputs, uses topology to constrain the search, and uses nucleotide-derived stem features to avoid purely topological degeneracy. The resulting output is an explicit partial map between stems, rather than only a nucleotide alignment, a string alignment, or a scalar distance. This partial map can be projected to a nucleotide-level alignment which we will use to demonstrate alignment quality.

# 3 Data Representation

In this section we define the representation used by our CoSTAR algorithm. The input is an RNA sequence together with a fixed secondary structure. The output of the preprocessing step is an ordered stem list, a reduced topology graph, a relation array over ordered pairs of stems, and a feature vector for each stem.

## 3.1 RNA sequence and secondary structure

An RNA sequence $S = S_1, \ldots, S_n$ is a string over the alphabet $\{A, C, G, U\}$. The indices $1, \ldots, n$ are ordered from the 5′ end to the 3′ end. For $1 \leq i \leq j \leq n$, let $S_{i,j} = S_i, \ldots, S_j$ denote the subsequence of $S$, and let $[i, j]$ denote the corresponding interval of nucleotide positions.

A base pair of $S$ is an ordered pair $(i, j)$, where $1 \leq i < j \leq n$. In a canonical sequence-compatible structure, $S_i$ and $S_j$ are complementary, i.e., $\{S_i, S_j\}$ is one of $\{A, U\}$, $\{C, G\}$, or $\{G, U\}$. The alignment algorithm below is specified for the base-pair set itself, so annotated non-canonical pairs can also be admitted provided that the pairing relation is well defined. A secondary structure $R$ for $S$ is a set of base pairs such that each nucleotide participates in at most one pair:

$$\{i, j\} \cap \{i', j'\} = \emptyset \quad \text{for all distinct } (i, j), (i', j') \in R.$$

A secondary structure is pseudoknot-free if it does not contain two base pairs $(i, j)$ and $(i', j')$ such that $i < i' < j < j'$ or $i' < i < j' < j$. Otherwise, the structure is pseudoknotted. Pseudoknotted structures are admissible inputs to our method.

## 3.2 Stems and bulge compression

A stem is a maximal chain of base pairs that are consecutive, or nearly consecutive after ignoring a bounded number of unpaired nucleotides. Let $\beta \geq 0$ be a user-defined bulge compression parameter. Consider two base pairs $(i, j)$ and $(k, \ell)$ in $R$, with $i < k < \ell < j$. They are stacked if $k = i + 1$ and $\ell = j - 1$. They are $\beta$-adjacent if all nucleotides in the intervals $[i + 1, k - 1]$ and $[\ell + 1, j - 1]$ are unpaired in $R$, and $(k - i - 1) + (j - \ell - 1) \leq \beta$. Thus, $\beta = 0$ recovers the ordinary definition of consecutive stacked base pairs. Larger values of $\beta$ allow short bulges or internal loops to be compressed when defining coarse stems. The original sequence and base-pair set are not changed; the parameter $\beta$ only determines which base pairs are grouped into the same stem. As default we set $\beta = 2$.

A stem $h$ is a maximal ordered sequence of base pairs $h = \big((i_1, j_1), (i_2, j_2), \ldots, (i_r, j_r)\big)$ such that $i_1 < i_2 < \cdots < i_r < j_r < \cdots < j_2 < j_1$, and $(i_{t-1}, j_{t-1})$ and $(i_t, j_t)$ are $\beta$-adjacent for every $1 \leq t \leq r$. The integer $r$ is the number of base pairs in the stem.

Each stem has two arms. The 5′ arm of $h$ is the interval $h^- = [i_1, i_r]$, and the 3′ arm is the interval $h^+ = [j_r, j_1]$. We refer to the ordered pair $(h^-, h^+)$ as the arm representation of $h$. The 5′ arm always precedes the 3′ arm in the sequence order.

Let $\mathcal{H}(R, \beta) = (h_1, \ldots, h_M)$ be the ordered stem list obtained from $R$ using compression parameter $\beta$, where stems are ordered by the left endpoint of their 5′ arms, and $M$ is the total number of stems. When $\beta$ is fixed by context, we write simply $\mathcal{H}(R)$.

## 3.3 Reduced topology graph

The reduced topology graph records the order of stem arms along the RNA backbone and the pairing relationship between the two arms of each stem. Let $V = \{h_i^-, h_i^+ : 1 \leq i \leq M\}$ be the set of arm intervals. Sort these $2M$ intervals by their left endpoint in the 5′-to-3′

order, and denote the resulting sequence by $v_1, v_2, \ldots, v_{2M}$. The reduced topology graph is $T_R = (V, E_{\text{bb}} \cup E_{\text{stem}})$, where $E_{\text{bb}} = \{(v_t, v_{t+1}) : 1 \le t < 2M\}$ is the set of directed backbone edges, and $E_{\text{stem}} = \{(h_i^-, h_i^+) : 1 \le i \le M\}$ is the set of stem edges. Unpaired regions that do not belong to compressed stems are not represented as vertices. Hence $T_R$ has $2M$ vertices, $2M-1$ backbone edges, and $M$ stem edges. This representation is related to descriptor-based RNA motif representations, where structures are described through ordered helices and intervening regions [23]; here we retain only the stem arms and their order because the alignment algorithm operates on stems rather than on unstructured intervals. The graph $T_R$ is used to define and visualize the reduced topology; the search algorithm uses the ordered stem list, the feature vectors, and the relation array induced by $T_R$.

## 3.4 Pairwise topological relations

The alignment algorithm uses pairwise topological relations between stems as constraints. We first define an ordering relation on arm intervals. For two arm intervals $x$ and $y$, write $x \prec y$ if interval $x$ ends before interval $y$ begins.

Let $A = (a, a')$ and $B = (b, b')$ be two distinct stems in arm representation, where $a$ and $b$ are the $5'$ arms and $a'$ and $b'$ are the $3'$ arms. We use the ordered relation-label tuple $\Lambda = (\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5, \lambda_6)$ defined by the following mutually exclusive cases:

| $\ell$ | condition | relation of $B$ with respect to $A$ | code alias |
|---|---|---|---|
| 1 | $b' \prec a$ | $5'$-disjoint | $5'$SL (stemloop) |
| 2 | $b \prec a \prec b' \prec a'$ | $5'$-crossing | $5'$PK (pseudoknot) |
| 3 | $b \prec a \prec a' \prec b'$ | contains | outer |
| 4 | $a \prec b \prec a' \prec b'$ | $3'$-crossing | $3'$PK |
| 5 | $a' \prec b$ | $3'$-disjoint | $3'$SL |
| 6 | $a \prec b \prec b' \prec a'$ | inside | inner. |

The semantic labels are used in the text. The code aliases are retained only for consistency with the implementation and figures.

For a structure $R$ with ordered stem list $\mathcal{H}(R) = (h_1, \ldots, h_M)$, define $\rho_R(i, j)$ as the unique label in $\Lambda$ giving the relation of stem $h_j$ with respect to stem $h_i$, for $i \neq j$. For any ordered pair of distinct stems, exactly one label in $\Lambda$ applies. The reciprocal relation is obtained by reversing the ordered pair. For example, if $\rho_R(i, j) = \lambda_1$, then $\rho_R(j, i) = \lambda_5$; if $\rho_R(i, j) = \lambda_3$, then $\rho_R(j, i) = \lambda_6$; and if $\rho_R(i, j) = \lambda_2$, then $\rho_R(j, i) = \lambda_4$.

We precompute $\rho_R$ for all ordered pairs of stems. The relation array for structure $R$ is the Boolean tensor $\Gamma_R \in \{0, 1\}^{M \times M \times 6}$, where for $i, j \in \{1, \ldots, M\}$ and $\ell \in \{1, \ldots, 6\}$,

$$\Gamma_R[i, j, \ell] = 1 \quad \Longleftrightarrow \quad i \neq j \text{ and } \rho_R(i, j) = \lambda_\ell.$$

Diagonal entries $\Gamma_R[i, i, \ell]$ are set to 0 and are not used. Since there are $M(M-1)$ ordered pairs of distinct stems, the relation array is computed in $O(M^2)$ time and stored in $O(M^2)$ space. This array is independent of the second structure being aligned, so it can be cached and reused across multiple alignments involving the same RNA structure.

## 3.5 Stem features

The reduced topology graph and relation array encode the relative arrangement of stems, but they do not distinguish stems with the same topological relation profile and different physical properties. Therefore each stem is also assigned a feature vector derived from the original nucleotide-level structure.

For a stem $h = \big((i_1, j_1), \ldots, (i_r, j_r)\big)$, we define three features. The first is the number of base pairs, $\phi_1(h) = r$. The second is the number of nucleotide positions between the two stem arms, $\phi_2(h) = j_r - i_r - 1$. This value provides geometric spacing considerations. The third is the magnitude of the stacking free energy contributed by adjacent stacked base-pair steps in the stem. Let $e_s(i, j)$ denote the nearest-neighbor stacking free energy for the adjacent stacked base-pair step $(i, j), (i + 1, j - 1)$, using standard thermodynamic parameters [17]. Then,

$$\phi_3(h) = - \sum_{\substack{1 \le t < r \\ i_{t+1} = i_t + 1,\ j_{t+1} = j_t - 1}} e_s(i_t, j_t).$$

If no adjacent stacked base-pair step occurs in $h$, then $\phi_3(h) = 0$. The negative value is used so that larger feature values correspond to larger stabilizing contributions.

The feature vector of $h$ is $\phi(h) = \big(\phi_1(h), \phi_2(h), \phi_3(h)\big)$. All feature vectors are computed once during preprocessing and are used later to define match and skip costs in the alignment algorithm.

### 3.6 Preprocessing output

For an input sequence–structure pair $(S, R)$ and bulge compression parameter $\beta$, the preprocessing step returns

$$\mathcal{D}(S, R, \beta) = \big(\mathcal{H}(R, \beta), T_R, \Gamma_R, \Phi_R\big),$$

where $\mathcal{H}(R, \beta)$ is the ordered stem list, $T_R$ is the reduced topology graph, $\Gamma_R$ is the relation array, and $\Phi_R = (\phi(h_1), \ldots, \phi(h_M))$ is the list of stem feature vectors. The alignment algorithm in the next section takes two such preprocessed representations as input.

## 4 CoSTAR Algorithm

Here we define the cost function and the algorithm that finds its minimum. Over the class of injective, direction-preserving, topologically consistent stem maps, the CoSTAR search is exact. Let $(S^F, R^F)$ and $(S^G, R^G)$ be two input sequence–structure pairs, and let

$$\mathcal{D}_F = \mathcal{D}(S^F, R^F, \beta) = (\mathcal{F}, T_F, \Gamma_F, \Phi_F) \quad \text{and} \quad \mathcal{D}_G = \mathcal{D}(S^G, R^G, \beta) = (\mathcal{G}, T_G, \Gamma_G, \Phi_G)$$

be their preprocessed representations. Here $\mathcal{F} = (F_1, \ldots, F_f)$ and $\mathcal{G} = (G_1, \ldots, G_g)$ are the ordered stem lists of the two structures.

### 4.1 Coarse alignments

A coarse alignment is a partial one-to-one correspondence between stems of $\mathcal{F}$ and stems of $\mathcal{G}$. Formally, let $I_F = \{1, \ldots, f\}$ and $I_G = \{1, \ldots, g\}$ be the stem-index sets of the two structures. A partial map

$$\mu : D \to I_G, \qquad D \subseteq I_F,$$

is injective if $\mu(i) \neq \mu(i')$ for all distinct $i, i' \in D$. We interpret $\mu(i) = j$ as matching stem $F_i$ to stem $G_j$. Stems in $I_F \setminus D$ and $I_G \setminus \mu(D)$ are skipped.

The search is restricted to direction-preserving maps. A partial map $\mu$ is direction-preserving if, for all $i, i' \in D$, $i < i' \implies \mu(i) < \mu(i')$. This constraint preserves the $5'$-to-$3'$ order of matched stems. It does not require the structures to be pseudoknot-free; pseudoknotted relationships are represented separately by the pairwise relation arrays.

A direction-preserving map is topologically consistent if every pair of matched stems has the same ordered relation in both structures. A partial map $\mu$ is topologically consistent if $\rho_F(p,i) = \rho_G(\mu(p),\mu(i))$ for every distinct $p, i \in D$. Equivalently, $\Gamma_F[p,i,\cdot] = \Gamma_G[\mu(p),\mu(i),\cdot]$ for every ordered pair of matched stems. The alignment problem is to find a minimum-cost partial map that is injective, direction-preserving, and topologically consistent.

## 4.2 Match and skip costs

The cost of matching $F_i$ to $G_j$ is the weighted $\ell_1$ distance between their feature vectors:

$$\delta_{ij} = \sum_{\ell=1}^{3} \omega_\ell \, |\phi_\ell(F_i) - \phi_\ell(G_j)|$$

where $\omega_1, \ldots, \omega_d \geq 0$ are feature weights (with default values of 1). All pairwise match costs are precomputed in an $f \times g$ matrix $\Delta = (\delta_{ij})$.

Skipping a stem is modeled as matching it to the empty stem. The skip costs are

$$\sigma_i^F = \sum_{\ell=1}^{3} \omega_\ell \phi_\ell(F_i), \qquad \sigma_j^G = \sum_{\ell=1}^{3} \omega_\ell \phi_\ell(G_j).$$

For a terminal coarse alignment represented by $\mu : D \to I_G$, the alignment cost is

$$C(\mu) = \sum_{i \in D} \delta_{i,\mu(i)} + \sum_{i \in I_F \setminus D} \sigma_i^F + \sum_{j \in I_G \setminus \mu(D)} \sigma_j^G.$$

Thus the objective is $\min_\mu C(\mu)$, where the minimum is taken over all injective, direction-preserving, and topologically consistent partial maps.

## 4.3 Search states

The algorithm searches over partial alignments in the 5′-to-3′ order of $\mathcal{F}$. A search node is a triple $u = (t, \mu, m)$, where $t \in \{1, \ldots, f\}$ is the number of stems of $\mathcal{F}$ whose status has already been decided, $\mu$ is a topologically consistent partial map with domain $D \subseteq \{1, \ldots, t\}$, and

$$m = \begin{cases} \max \mu(D), & D \neq \emptyset, \\ 0, & D = \emptyset. \end{cases}$$

The value $m$ is the largest index of a stem in $\mathcal{G}$ that has been matched so far. By directionality, any unmatched stem $G_j$ with $j < m$ is already skipped and cannot be used later.

Equivalently, a node can be represented by two alignment arrays. The $\mathcal{F}$-array has length $f$ and stores, for each $i$, either the matched index $\mu(i)$, a skip symbol $\perp$, or an undecided symbol $*$. The $\mathcal{G}$-array is defined analogously. In the implementation these two symbols are encoded by sentinel values; the mathematical description uses $\perp$ and $*$.

For a node $u = (t, \mu, m)$, define the fixed cost

$$C_{\text{fix}}(u) = \sum_{i \in D} \delta_{i,\mu(i)} + \sum_{i \in \{1,\ldots,t\} \setminus D} \sigma_i^F + \sum_{j \in \{1,\ldots,m\} \setminus \mu(D)} \sigma_j^G.$$

This is the cost of all decisions that can no longer change.

The remaining undecided stems are

$$U_F(u) = \{t+1, \ldots, f\}, \qquad U_G(u) = \{m+1, \ldots, g\}.$$

To prioritize the search, we use a lower bound on the cost of completing a partial alignment. For $i \in U_F(u)$, define

$$\eta_F(i;u) = \min\left\{\sigma_i^F, \min_{j\in U_G(u)} \delta_{ij}\right\},$$

and for $j \in U_G(u)$, define

$$\eta_G(j;u) = \min\left\{\sigma_j^G, \min_{i\in U_F(u)} \delta_{ij}\right\}.$$

The node key is

$$L(u) = C_{\text{fix}}(u) + \max\left\{\sum_{i\in U_F(u)} \eta_F(i;u), \sum_{j\in U_G(u)} \eta_G(j;u)\right\}.$$

The two sums are not added, since doing so would double-count potential future matches. Their maximum is still a valid lower bound and is sufficient for branch-and-bound pruning.

▶ **Lemma 1.** *For every search node $u$, $L(u)$ is a lower bound on the cost of every terminal coarse alignment extending $u$.*

**Proof.** The term $C_{\text{fix}}(u)$ is the exact cost of all matched and skipped stems whose status has already been decided. Consider any terminal extension of $u$. Each remaining stem $F_i \in U_F(u)$ is either skipped, contributing $\sigma_i^F$, or matched to some remaining stem $G_j$, contributing $\delta_{ij}$. Hence its contribution is at least $\eta_F(i;u)$, and the remaining cost is at least $\sum_{i\in U_F(u)} \eta_F(i;u)$. The same argument applied to the remaining stems of $\mathcal{G}$ shows that the remaining cost is also at least $\sum_{j\in U_G(u)} \eta_G(j;u)$. Therefore it is at least the maximum of these two lower bounds. Adding $C_{\text{fix}}(u)$ gives $L(u)$. ◀

## 4.4 Node expansion

Let $u = (t, \mu, m)$ be a non-terminal node, so $t < f$. The next stem to be decided is $F_i$ with $i = t + 1$. There is always a skip child $(t + 1, \mu, m)$, whose fixed cost increases by $\sigma_i^F$.

A match child is generated for a candidate $G_j$, with $j > m$, only if the candidate satisfies all topological constraints induced by the stems already matched in $\mu$. Specifically, $F_i \mapsto G_j$ is admissible if

$$\Gamma_F[p, i, \cdot] = \Gamma_G[\mu(p), j, \cdot] \qquad \forall p \in D.$$

If this condition holds, the child node is

$$(t + 1, \mu \cup \{i \mapsto j\}, j).$$

The fixed cost of this child increases by the match cost $\delta_{ij}$ and by the skip costs of any stems of $\mathcal{G}$ that are passed over:

$$\delta_{ij} + \sum_{q=m+1}^{j-1} \sigma_q^G.$$

The skipped stems $G_{m+1}, \ldots, G_{j-1}$ cannot be used later because of the direction-preserving constraint.

Since every pair of matched stems is checked when the later stem in $\mathcal{F}$ is added, every terminal node produced by this expansion rule is topologically consistent.

## 4.5 Alignment tree and initialization

The search space is a rooted tree. The root is $u_0 = (0, \emptyset, 0)$. A root-to-leaf path decides the status of each stem $F_1, \ldots, F_f$ exactly once. When a leaf has $t = f$, all remaining unmatched stems $G_{m+1}, \ldots, G_g$ are skipped. Therefore the terminal cost of a leaf $u = (f, \mu, m)$ is

$$C_{\text{term}}(u) = C_{\text{fix}}(u) + \sum_{j=m+1}^{g} \sigma_j^G.$$

This is equal to the terminal alignment cost $C(\mu)$.

For implementation, we use a shortcut initialization rather than generating leading skip decisions one at a time. The root children are all possible first matched pairs $F_i \mapsto G_j$, with $1 \le i \le f$ and $1 \le j \le g$. In the child corresponding to $F_i \mapsto G_j$, stems $F_1, \ldots, F_{i-1}$ and $G_1, \ldots, G_{j-1}$ are immediately fixed as skipped. Subsequent expansion then proceeds from $F_{i+1}$ and considers only stems of $\mathcal{G}$ with index larger than $j$. Thus the shortcut suppresses paths whose only purpose is to choose the first matched pair after a prefix of skipped stems.

This initialization restricts the implemented search to nonempty partial stem maps: every generated terminal alignment contains at least one matched pair. The all-skip map $\mu = \emptyset$, with cost $C(\emptyset) = \sum_{i=1}^{f} \sigma_i^F + \sum_{j=1}^{g} \sigma_j^G$, is therefore not generated by the shortcut initialization.

## 4.6 Branch-and-bound traversal

The traversal procedure is given in Algorithm 1. The priority queue stores active search nodes ordered by their lower-bound key $L(u)$. The variable $C^*$ stores the best terminal alignment cost found so far, and $\mathcal{S}^*$ stores all terminal alignments attaining this cost. A node is discarded whenever its lower bound exceeds $C^*$, since no completion of that node can improve the current best solution. To enumerate all optimal alignments, nodes with lower bound equal to $C^*$ are retained until they are either expanded or shown not to lead to another terminal alignment of cost $C^*$.

If only one optimal alignment is required, the traversal may stop once a terminal coarse alignment of cost $C^*$ has been found and every active node has lower bound at least $C^*$. To enumerate all optimal alignments, the search continues to expand nodes with lower bound equal to $C^*$, and records every terminal alignment with terminal cost $C^*$.

▶ **Theorem 2.** *The branch-and-bound traversal returns a minimum-cost alignment over the set of injective, direction-preserving, and topologically consistent stem maps.*

**Proof.** The expansion rule enumerates exactly the direction-preserving choices for each stem $F_i$: either $F_i$ is skipped, or it is matched to a later unused stem of $\mathcal{G}$. A match child is generated only when its relation-array entries agree with all previously matched stems, so every terminal node is topologically consistent. Conversely, any direction-preserving and topologically consistent complete map determines a unique root-to-leaf path in the tree.

By Lemma 1, $L(u)$ is a lower bound on the cost of every terminal coarse alignment extending $u$. Therefore, when a node has $L(u) > C^*$, no descendant of that node can improve the best terminal coarse alignment already found. The traversal only prunes such nodes. Hence no alignment with cost smaller than $C^*$ is discarded. When the queue is empty, or when all remaining nodes have lower bound greater than $C^*$, no unexplored completion can have cost below $C^*$. Thus $C^*$ is the optimal alignment cost. ◀

**Algorithm 1** Best-first branch-and-bound traversal of the alignment tree.

**Require:** Root node $u_0$, lower-bound function $L(\cdot)$, terminal-cost function $C_{\text{term}}(\cdot)$
**Ensure:** Set $\mathcal{S}^*$ of minimum-cost coarse alignments and optimal cost $C^*$

$Q \leftarrow$ empty priority queue ordered by increasing $L(\cdot)$
$C^* \leftarrow +\infty$
$\mathcal{S}^* \leftarrow \emptyset$
insert $u_0$ into $Q$ with key $L(u_0)$
**while** $Q \neq \emptyset$ and $\min_{u \in Q} L(u) \leq C^*$ **do**
    $u \leftarrow \textsc{ExtractMin}(Q)$
    **if** $L(u) > C^*$ **then**
        **continue**
    **end if**
    **if** $u$ is terminal **then**
        $c \leftarrow C_{\text{term}}(u)$
        **if** $c < C^*$ **then**
            $C^* \leftarrow c$
            $\mathcal{S}^* \leftarrow \{\mu_u\}$
        **else if** $c = C^*$ **then**
            $\mathcal{S}^* \leftarrow \mathcal{S}^* \cup \{\mu_u\}$
        **end if**
    **else**
        **for all** $v \in \textsc{Expand}(u)$ **do**
            **if** $L(v) \leq C^*$ **then**
                insert $v$ into $Q$ with key $L(v)$
            **end if**
        **end for**
    **end if**
**end while**
**return** $(C^*, \mathcal{S}^*)$

## 4.7 Example of topologically constrained expansion

Figure 1 illustrates one expansion step. For a partial alignment with matched-domain $D$, the next stem $F_i$ defines a constraint vector $\big(\Gamma_F[p, i, \cdot]\big)_{p \in D}$. For each candidate $G_j$, the algorithm constructs the corresponding vector $\big(\Gamma_G[\mu(p), j, \cdot]\big)_{p \in D}$. The candidate $G_j$ is accepted exactly when these two vectors are identical. Otherwise, matching $F_i$ to $G_j$ would change at least one previously established topological relation, and the child is not generated. The skip child for $F_i$ is always generated.

# 5 Experimental Setup

Table 1 summarizes the Rfam families used in the benchmark and the range of sequence lengths and stem counts after preprocessing.

## 5.1 Benchmark instances

For each Rfam family, we selected the first ten sequences and extracted the consensus structure, converting it from WUSS notation into dotbracket and applied the preprocessing procedure described in Section 3. The resulting structures from the same family were not

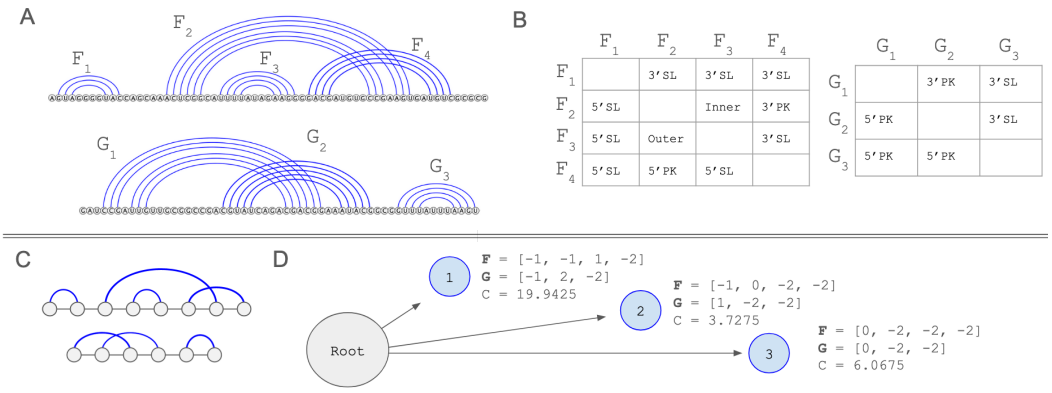



**Figure 1** Example of topologically constrained expansion. **A**) Two RNA structures $F$ and $G$ as arc diagrams (each arc connects two ends of a base pair) with stems labelled. **B**) Relation-array summaries. Rows give the reference stem, and columns give the relation of the column stem with respect to the row stem. Reciprocal entries contain reciprocal relation labels. **C**) Reduced topology representations of $F$ and $G$ used by the alignment algorithm. **D**) Example search-tree nodes. Each node stores the current partial stem map, skipped stems, undecided stems, and its lower-bound key. A candidate child is generated only if its relation-array constraints match those induced by the previously matched stems.

necessarily identical at the stem level: insertions, deletions, and small bulges can change how base pairs are grouped into stems under the chosen bulge compression parameter. We then constructed cross-family alignment instances. For each unordered pair of distinct Rfam families, we formed all $10 \times 10$ cross-family comparisons by aligning each selected structure from one family with each selected structure from the other. This produced $\binom{7}{2} \times 10 \times 10 = 2100$ cross-family pairwise alignment instances. We used cross-family rather than within-family pairs because they give a more stringent test of the search procedure: stem features, stem counts, and pairwise topological relations are expected to be less conserved across families.

## 5.2 Measured quantities

For a pair of preprocessed inputs $\mathcal{D}_F = (\mathcal{F}, T_F, \Gamma_F, \Phi_F)$, $\mathcal{D}_G = (\mathcal{G}, T_G, \Gamma_G, \Phi_G)$, let $\mathcal{F} = (F_1, \ldots, F_f)$ and $\mathcal{G} = (G_1, \ldots, G_g)$. We use $I(\mathcal{F}, \mathcal{G}) = fg$ as a simple proxy for the number of initial match choices in the shortcut initialization of the alignment tree. This quantity does not determine the full search-tree size, but it captures the first-order dependence on the number of stems in the two inputs.

For each ordered input pair, let $\mathcal{T}_{\mathcal{F},\mathcal{G}}$ be the alignment tree generated by CoSTAR. We recorded the number of generated search nodes and the maximum tree width

$$W(\mathcal{F}, \mathcal{G}) = \max_d \left| \{ u \in \mathcal{T}_{\mathcal{F},\mathcal{G}} : \mathrm{depth}(u) = d \} \right| .$$

Thus $W(\mathcal{F}, \mathcal{G})$ is the largest number of partial alignments present at any search depth. To measure the size of the relation-consistent search space rather than the behavior of a particular branch-and-bound run, we disabled only the cost-based pruning rule $L(u) > C^*$ for these measurements. The directionality constraint and the relation-array constraints remained active. Consequently, the reported node counts and tree widths measure the search tree induced by valid topological extensions.

**Table 1** Rfam families used in the benchmark and their preprocessing statistics. The first column represent the Rfam family ID; the second column (N. Seq) presents the number of sequences in that family; the third column (PK) identifies whether the family contains a pseudoknot. We use 10 samples from each family and the following columns characterize those selected samples: the fourth column (Seq. Id.) identifies the sequence similarity in the family subset; the fifth and the sixth columns present the minimum and maximum lengths of sequences in the family subset; and the last two columns present the minimum and maximum number of stems in the family subset as identified in the Rfam structure.

| Rfam family | N. Seq. | PK | Seq. Id. | Min. NT | Max. NT | Min. stems | Max. stems |
|---|---|---|---|---|---|---|---|
| RF00001 | 712 | No | 0.92 | 116 | 118 | 5 | 6 |
| RF00003 | 100 | No | 0.87 | 162 | 164 | 5 | 5 |
| RF00008 | 85 | No | 0.57 | 40 | 82 | 3 | 4 |
| RF00165 | 52 | Yes | 0.71 | 61 | 63 | 2 | 2 |
| RF00174 | 434 | Yes | 0.72 | 180 | 224 | 12 | 15 |
| RF00381 | 13 | Yes | 0.85 | 56 | 59 | 2 | 2 |
| RF00507 | 51 | Yes | 0.69 | 78 | 81 | 3 | 4 |

## 6 Results

We evaluated the coarse alignment algorithm on the Rfam benchmark described in Section 5.

**Search-tree growth.** Figure 2 summarizes the dependence of running time and tree width on stem-level input size. The left panel plots running time as a function of $I(\mathcal{F}, \mathcal{G}) = fg$. Instances with larger $I(\mathcal{F}, \mathcal{G})$ tend to induce wider search trees and longer running times. This is consistent with the structure of the algorithm: after preprocessing, the dominant cost is not sequence length, but the number of partial stem maps that survive the directionality and topological consistency constraints. The dependence is not solely a function of $f$ and $g$, since two pairs with the same stem counts can have different relation arrays and therefore different numbers of admissible extensions. For this panel, the search-driving structure $\mathcal{F}$ is set as the structure with more stems.

**Effect of input ordering.** The alignment problem is symmetric in its final objective, but the search tree is not symmetric in the ordered inputs. The algorithm decides stems of $\mathcal{F}$ sequentially and branches over admissible unused stems of $\mathcal{G}$. Thus, placing the structure with more stems in the search-driving role $\mathcal{F}$ increases the depth of the tree but decreases the number of candidate targets considered at each expansion. In the benchmark instances, this ordering reduced maximum tree width. The right panel of Figure 2 compares the alignment tree width and alignment time to calculate $\mathcal{F} \to \mathcal{G}$ with the width and time for $\mathcal{G} \to \mathcal{F}$, where $\mathcal{F}$ is the structure with more stems. Color indicates the number of stems in $\mathcal{F}$ minus the number in $\mathcal{G}$. When the search-driving structure has more stems, the tree width tends to be lower.

**Preprocessing cost.** The non-search preprocessing steps are comparatively inexpensive. For a structure with $n$ nucleotides, $p$ base pairs, and $M$ stems, parsing the structure and computing stem features are linear in the nucleotide-level representation after the base pairs have been ordered. The relation array is computed by comparing all ordered pairs of stems, requiring $O(M^2)$ time and $O(M^2)$ space. Since the relation array $\Gamma_R$ and feature list $\Phi_R$ depend only on a single input structure, they can be cached and reused when the same RNA participates in multiple pairwise alignments.

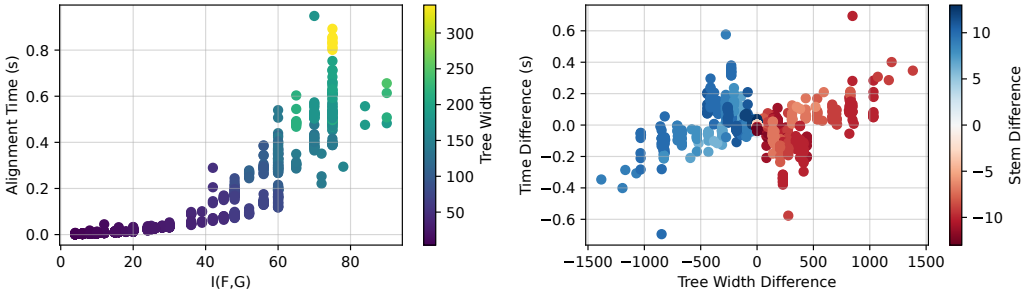



**Figure 2** Empirical search behavior on 2100 cross-family Rfam alignment instances. **Left:** Alignment time as a function of the initial stem-pair count $I(\mathcal{F}, \mathcal{G}) = fg$. Points are colored by maximum tree width $W(\mathcal{F}, \mathcal{G})$, computed with cost-based pruning disabled but with directionality and topological constraints enforced. **Right:** Effect of input structure ordering on alignment tree. One structure is set as the search-driving structure ($\mathcal{F} \to \mathcal{G}$) and tree width and alignment time are calculated. The reciprocal ($\mathcal{G} \to \mathcal{F}$) is also evaluated. For a pair of structures we plot two points showing the change in tree width and alignment time relative to the alternative structure ordering. Color shows which structure has more stems where positive values (blue) indicates the search-driving structure has more stems.

The alignment tree is the main source of computational cost. If relation-array constraints are ignored but directionality is retained, the number of complete direction-preserving partial maps is $\sum_{k=0}^{\min(f,g)} \binom{f}{k}\binom{g}{k}$, because choosing $k$ stems from $\mathcal{F}$ and $k$ stems from $\mathcal{G}$ determines a unique order-preserving matching. The generated search tree also contains prefixes of such maps. Relation-array constraints reduce this space in an input-dependent way by rejecting candidate matches that would alter the pairwise topology of already matched stems.

**Adversarial structures.** The topological constraints are least effective when many stems have indistinguishable relation profiles. For example, if both inputs contain many repeated stems that are mutually disjoint along the backbone, then most pairwise relations are only $5'$-disjoint or $3'$-disjoint. In such cases, relation-array constraints reject few candidate extensions, and the search approaches the directionality-only regime.

As a stress test, we aligned two structures with 11 stems each whose relation patterns were largely repetitive and disjoint. With cost-based pruning disabled, the alignment required approximately six minutes in our implementation, generated about $10^6$ search nodes, and reached a maximum tree width of approximately $2.8 \times 10^5$. This instance illustrates the data-dependent nature of the search: the number of stems determines the broad scale of the search tree, but the pairwise topology of those stems determines how much pruning is obtained from the relation-array constraints.

## 6.1 Qualitative comparison with LaRA

We first compare CoSTAR with LaRA, the closest available method for aligning RNA sequences with pseudoknotted secondary structures. LaRA optimizes a nucleotide-level sequence–structure alignment, whereas CoSTAR optimizes a partial map between stems. Thus, the two methods do not return the same type of object. We outline a simple projection method in appendix A that converts our mapping into a nucleotide-level alignment. The examples below show that, even when LaRA is run in a structure-prioritized setting, its nucleotide-level alignment can fail to recover the stem correspondences required for coarse structural comparison.

**Synthetic pseudoknotted pair.** We constructed a synthetic pair in which the two RNAs share a pseudoknotted structural core but differ in flanking and additional stems. This design tests whether an aligner can ignore non-corresponding structural elements and align the shared pseudoknotted block. Listing 1 shows the LaRA alignment. The shared paired regions are not aligned as coherent stem blocks: gaps are inserted within paired regions, and the pseudoknotted arms are fragmented across the alignment. As a result, the output does not provide a usable stem-to-stem correspondence for this example.

**Listing 1** LaRA nucleotide-level alignment of the synthetic pseudoknotted pair.

```
F LaRA
--AGUAGCUCGGCCAUUUUAGAGUCGGACGAUUUGUGGCGAAGUGUGUCGGCGCGAUGCUUGACAGUUCGG----AUGAUCCGAUUG
__........(((.((((.....[[[.[[..........)))..))))..............]].]]]____...........
G LaRA
CAGACAGGUUUAACGA-------GCGGUU--UUUGAGUCAUCACAUGCCUAAACGAUGCUGGGCUCCGCGGCGACUCGCUCUACCGA
.(((....))).....(_______((((((__.....[[[.[[....[[[[.)))..))))]]]]...([]].]]]..)).......
```

Listing 2 shows the nucleotide-level projection obtained from the CoSTAR stem map on the same input pair. Vertical bars mark the boundaries induced by the coarse alignment. The shared pseudoknotted arms are aligned within corresponding blocks, while additional stems and unmatched regions are separated by gaps or distinct blocks. This illustrates the intended use of CoSTAR: the structural correspondence is determined at the stem level before nucleotide columns are introduced.

**Listing 2** CoSTAR nucleotide-level projection of the synthetic pseudoknotted pair. Vertical bars mark blocks induced by the coarse stem map.

```
F CoSTAR
|AGUAGCUC_______|GGCCAUUU|UAGAG|UCGGAC|GAUUUGUGGCG|AAGUGUGUC|GGCGCGAUGCUUGACA|GUUCGG|AUGAUCCGAUUG|
|........_______|(((.((((|.....|[[[.[[|...........|)))..))))|................|]].]]]|...........|
G CoSTAR
|CAGACAGGUUUAACG|AGC_GGUU|UUUGA|GUCAUC|ACAUGCCUA__|AACGAUGCU|GGGCUCCGC_______|GGCGAC|UCGCUCUACCGA|
|.(((....))).....|(((_((((|.....|[[[.[[|....[[[[.__|)))..))))|]]]].((_______|]].]]]|..)).........|
```

**Stem-deletion perturbation.** We next tested whether the methods can align a structure to a perturbed copy with a deleted stem. We selected a structure from Rfam family RF00507 and created a knockout by removing the first stem together with external unstructured nucleotides. The desired coarse alignment should skip the deleted 5′ stem and align the remaining stems after a leading gap.

Listing 3 shows the alignment produced by LaRA. LaRA introduces a large 5′ gap, but the gap is placed within the remaining paired region rather than before the corresponding structural block. Consequently, the deleted stem is not represented as a clean skipped structural element, and the alignment does not recover the intended stem-level offset.

**Listing 3** LaRA nucleotide-level alignment of the RF00507 stem-deletion perturbation.

```
Reference LaRA
GGGUUCGGGGUACUAGUGUGAAUGCCCGGCUAGUACCCUGUGCUAGUGGUUUAUCUACUGAUGUUCAAUUAAGGGCAUUUG
.......((((((((((......[[[[[..)))))))))...(.((...((......))...)).).......]]]]]....
Knockout LaRA
UG--------------------C--CCGGCUAGUACCCUGUGCUAGUGGUUUAUCUACUGAUGUUCAAUUAAGGGCA----
((____________________(__((.............(.((...((......))...)).).......)))))____
```

CoSTAR instead represents the deletion as a skipped leading block and aligns the remaining stems directly, as shown in Listing 4. This is the behavior required for the coarse alignment task: the perturbation is localized to the deleted stem, while the rest of the stem map remains interpretable.

**Listing 4** CoSTAR nucleotide-level projection of the RF00507 stem-deletion perturbation. The deleted leading stem is represented as a skipped block.

```
Reference CoSTAR
|GGGUUCGGGGUACUAGUGUGAA|UGCCC|GGCUAGUACCCUGU|GCUA|GUG|GU|UUAUCU|AC|UGA|UGUU|CAAUUAA|GGGCA|UUUG|
|.......((((((((((......|[[[[[|..)))))))))...|(.(((|...|(((|......|))|...|)).)|.......|]]]]]|....|
Knockout CoSTAR
|_____________________|UGCCC|GGCUAGUACCCUGU|GCUA|GUG|GU|UUAUCU|AC|UGA|UGUU|CAAUUAA|GGGCA|____|
|_____________________|(((((|.............|(.(((|...|(((|......|))|...|)).)|.......|)))))|____|
```

Together, these examples demonstrate the distinction between the two objectives. LaRA can align pseudoknotted RNA at nucleotide resolution, but the resulting alignment need not preserve the stem-level correspondences needed for coarse structural comparison. CoSTAR directly optimizes that stem-level correspondence and can then project the resulting map to nucleotide resolution.

## 7 Discussion and Future Work

We presented CoSTAR, a coarse alignment algorithm for RNA secondary structures that may contain pseudoknots. CoSTAR represents each input by its ordered stems, pairwise stem relations, and nucleotide-derived stem features, and returns a partial injective map between stems. The method therefore lies between topology-only representations, such as RNA abstract shapes and RNA-as-Graph [13, 11], and nucleotide-level sequence–structure alignment methods such as LaRA 2 [28]. Its main computational advantage is that the dominant input size is the number of stems, not the number of nucleotides: stem features are computed once, relation arrays require $O(M^2)$ time and space for $M$ stems, and these precomputed objects can be cached across alignments.

The search over partial stem maps remains the computational bottleneck and can grow exponentially in the number of stems in the worst case. The empirical results show, however, that directionality and relation-array constraints reduce the explored search space on the benchmark instances. Because the search is asymmetric in its ordered inputs, using the structure with more stems as the search-driving input $\mathcal{F}$ typically reduces tree width, as observed in the reciprocal-orientation experiments. The least favorable cases are structures with many repeated, mutually disjoint stems, where most relations are $5'$-disjoint or $3'$-disjoint and topological constraints reject few candidates.

Several modeling choices can be refined. The current lower bound is admissible but relaxed: it allows each remaining stem to choose its cheapest match or skip cost without enforcing future injectivity or topological consistency. Tighter admissible bounds, such as a minimum-cost bipartite matching over the remaining stems with skip vertices, could reduce the number of expanded nodes. The cost function is also intentionally simple, using weighted distances between stem features and corresponding skip costs. Since RNA similarity is task-dependent [3], future scoring models could incorporate sequence composition, covariation, learned feature weights, or stronger penalties for disrupting pseudoknotted topology.

CoSTAR can also be used as an anchoring step for nucleotide-level alignment. If the coarse map matches $F_i$ to $G_j$, then the nucleotide positions in the arms of $F_i$ can be constrained or encouraged to align to the corresponding arms of $G_j$, and intervals between matched stems can be solved as smaller sequence–structure alignment subproblems. We provide a simple nucleotide-projection method, though future work should evaluate this projection on curated homologous RNAs, viral pseudoknot benchmarks, and comparisons against nucleotide-level aligners and topology-based distance methods. Additional extensions include studying the effect of the bulge-compression parameter $\beta$, which trades off fewer stems and faster search against the risk of merging stems that should remain distinct.

## References


1 Markus J. Bauer, Gunnar W. Klau, and Knut Reinert. Accurate multiple sequence-structure alignment of RNA sequences using combinatorial optimization. *BMC Bioinformatics*, 8:271, 2007. doi:10.1186/1471-2105-8-271.

2 Ian Brierley, Simon Pennell, and Robert J. C. Gilbert. Viral RNA pseudoknots: versatile motifs in gene expression and replication. *Nature Reviews. Microbiology*, 5:598–610, 2007.

3 James W. Brown, Amanda Birmingham, Paul E. Griffiths, Fabrice Jossinet, Rym Kachouri-Lafond, Rob Knight, B. Franz Lang, Neocles Leontis, Gerhard Steger, Jesse Stombaugh, and Eric Westhof. The RNA structure alignment ontology. *RNA*, 15(9):1623–1631, 2009. doi:10.1261/rna.1601409.

4 Thomas R. Cech and Joan A. Steitz. The noncoding RNA revolution—trashing old rules to forge new ones. *Cell*, 157(1):77–94, 2014. doi:10.1016/j.cell.2014.03.008.

5 José Almeida Cruz and Eric Westhof. The dynamic landscapes of RNA architecture. *Cell*, 136(4):604–609, 2009. doi:10.1016/j.cell.2009.02.003.

6 Padideh Danaee, Mason N. Rouches, Michelle Wiley, Dezhong Deng, Liang Huang, and David A. Hendrix. bpRNA: large-scale automated annotation and analysis of RNA secondary structure. *Nucleic Acids Research*, 46:5381–5394, 2018.

7 Jennifer A. Doudna and Thomas R. Cech. The chemical repertoire of natural ribozymes. *Nature*, 418(6894):222–228, 2002. doi:10.1038/418222a.

8 Sean R. Eddy. Computational analysis of conserved RNA secondary structure in transcriptomes and genomes. *Annual Review of Biophysics*, 43:433–456, 2014. doi:10.1146/annurev-biophys-051013-022950.

9 Sean R. Eddy and Richard Durbin. RNA sequence analysis using covariance models. *Nucleic Acids Research*, 22(11):2079–2088, 1994. doi:10.1093/nar/22.11.2079.

10 Stéfan Engelen and Fariza Tahi. Predicting RNA secondary structure by the comparative approach: how to select the homologous sequences. *BMC Bioinformatics*, 8:464, 2007. doi:10.1186/1471-2105-8-464.

11 Daniela Fera, Namhee Kim, Nir Shiffeldrim, Joshua Zorn, Uri Laserson, Hin Hark Gan, and Tamar Schlick. RAG: RNA-As-Graphs web resource. *BMC Bioinformatics*, 5:88, 2004. doi:10.1186/1471-2105-5-88.

12 Laura R. Ganser, Megan L. Kelly, Daniel Herschlag, and Hashim M. Al-Hashimi. The roles of structural dynamics in the cellular functions of RNAs. *Nature Reviews Molecular Cell Biology*, 20(8):474–489, 2019. doi:10.1038/s41580-019-0136-0.

13 Robert Giegerich, Björn Voß, and Marc Rehmsmeier. Abstract shapes of RNA. *Nucleic acids research*, 32 16:4843–51, 2004.

14 Jean-Pierre Séhi Glouzon, Jean-Pierre Perreault, and Shengrui Wang. The super-n-motifs model: a novel alignment-free approach for representing and comparing RNA secondary structures. *Bioinformatics*, 33:1169–1178, 2017.

15 Matthias Höchsmann, Thomas Töller, Robert Giegerich, and Stefan Kurtz. Local similarity in RNAsecondary structures. *Computational Systems Bioinformatics. CSB2003. Proceedings of the 2003 IEEE Bioinformatics Conference. CSB2003*, pages 159–168, 2003. doi:10.1109/CSB.2003.1227315.

16 Brittany Lasher and David A. Hendrix. bpRNA-align: improved RNA secondary structure global alignment for comparing and clustering RNA structures. *RNA*, 29(5):584–595, 2023. doi:10.1261/rna.079211.122.

17 David H. Mathews, Jeffrey Sabina, Michael Zuker, and Douglas H. Turner. Expanded sequence dependence of thermodynamic parameters improves prediction of RNA secondary structure. *Journal of molecular biology*, 288 5:911–40, 1999.

18 Eugenio Mattei, Gabriele Ausiello, Fabrizio Ferrè, and Manuela Helmer-Citterich. A novel approach to represent and compare RNA secondary structures. *Nucleic Acids Research*, 42:6146–6157, 2014.

19 John S. Mattick and Igor V. Makunin. Non-coding RNA. *Human Molecular Genetics*, 15(suppl_1):R17–R29, 2006. doi:10.1093/hmg/ddl046.

20 Kevin V. Morris and John S. Mattick. The rise of regulatory RNA. *Nature Reviews Genetics*, 15(6):423–437, 2014. doi:10.1038/nrg3722.

21 Kayla M. Peck and Adam S. Lauring. Complexities of viral mutation rates. *Journal of Virology*, 92, 2018.

22 Michela Quadrini, Luca Tesei, and Emanuela Merelli. ASPRAlign: a tool for the alignment of RNA secondary structures with arbitrary pseudoknots. *Bioinformatics*, 36(11):3578–3579, 2020. doi:10.1093/bioinformatics/btaa147.

23 Ladislav Rampásek, Randi M. Jimenez, Andrej Lupták, Tomás Vinař, and Broňa Brejová. RNA motif search with data-driven element ordering. *BMC Bioinformatics*, 17, 2016.

24 David Sankoff. Simultaneous solution of the RNA folding, alignment and protosequence problems. *SIAM Journal on Applied Mathematics*, 45(5):810–825, 1985. doi:10.1137/0145048.

25 Bruce A. Shapiro and Kaizhong Zhang. Comparing multiple RNA secondary structures using tree comparisons. *Bioinformatics*, 6(4):309–318, 1990. doi:10.1093/bioinformatics/6.4.309.

26 Ignacio Jr. Tinoco and Carlos Bustamante. How RNA folds. *Journal of Molecular Biology*, 293(2):271–281, 1999. doi:10.1006/jmbi.1999.3001.

27 Sebastian Will, Kristin Reiche, Ivo L. Hofacker, Peter F. Stadler, and Rolf Backofen. Inferring noncoding RNA families and classes by means of genome-scale structure-based clustering. *PLoS Computational Biology*, 3(4):e65, 2007. doi:10.1371/journal.pcbi.0030065.

28 Jörg Winkler, Gianvito Urgese, Elisa Ficarra, and Knut Reinert. LaRA 2: parallel and vectorized program for sequence–structure alignment of RNA sequences. *BMC Bioinformatics*, 23, 2022. doi:10.1186/S12859-021-04532-7.


## A Postprocessing Projection

An intuitive way to represent the output of our stem mapping is by projecting the mapping back to a nucleotide-level alignment. The alignment is broken up into segments of mapped and unmapped regions. Segments are separated by pipes for visual convenience. Unmapped regions are those that contain either no structure, or contain structure that was not included in the mapping. An example of structure not part of the mapping can be observed in Listing 2 where an unmapped loop in the G structure is found in the most $5'$ section. By construction there are an odd number of segments in a projection, alternating between unmapped and mapped, with both $5'$ and $3'$ extremes being unmapped.

When introducing spacer symbols to the two inputs, we follow two principles: 1) in an unmapped section, spacers are added to the shorter of the two inputs on the $3'$ side equal to the difference; 2) in mapped sections, spacers fill internal loops and bulges first, then additional spacers are added to the $3'$ side. Since our focus is on structures, we do not further refine the nucleotide sequences of the unmapped segments, which could be performed by merging gaps with adjacent segments or applying existing pairwise sequence alignment tools.

## B Examples of Nucleotide-Level Alignment Behavior

LaRA 2 was used as a point of comparison because it supports alignment of RNA sequences with pseudoknotted secondary structures [28]. However, LaRA solves a nucleotide-level sequence–structure alignment problem, whereas our method computes a stem-level structural map. The two outputs are therefore not identical objects. The examples below illustrate cases in which a nucleotide-level alignment can be difficult to interpret when the intended comparison is a coarse stem correspondence.

Let $S^F, R^F$ and $S^G, R^G$ denote the two input sequence–structure pairs. A nucleotide-level aligner returns aligned strings $\widetilde{S}^F, \widetilde{R}^F, \widetilde{S}^G, \widetilde{R}^G$, where gap symbols are inserted into the sequence and structure strings. A base pair $(i, j) \in R^F$ is preserved by the alignment if the columns containing $S^F_i$ and $S^F_j$ are aligned to two non-gap nucleotides $S^G_p$ and $S^G_q$ such that $(p, q) \in R^G$. In the examples below, we use this criterion only as a qualitative diagnostic: the purpose is not to benchmark LaRA, but to show why a stem-level output can be more appropriate for some structural comparison tasks.

LaRA has three relevant parameters in these examples. The parameter `balance` controls the relative contribution of sequence and structure. We set `balance=INF` to prioritize structural agreement. The parameters `gapOpen` and `gapExtend` define the affine cost for opening and extending gaps, respectively.

**Default gap parameters.** The first run used the two structures shown in Figure 1, with `balance=INF` and the default LaRA gap parameters. Lines labelled $F$ and $G$ give the aligned sequence and structure strings for the two inputs.

**Listing 5** LaRA output with `balance=INF` and default gap parameters.

```
F sequence:  AGUAGGGGUACCAGCAAACUCGGCAUUUUAUAGAAGGGGACGAUGUGCCGAAGUGAUGUCGCGCG
F structure: .(((....))).....((((((..((((....)))).[[[[.....))).))))..]]]].....
G sequence:  GAUCCGAUUGUU_GCGGCCGACGUAUCAGAC_GACGGAAAUACGGCGGUUUAUUUA______AGU
G structure: ...(((.(((..._........[[[[[....)_)))))..]]]]]...((((....)______)))
```

In this output, gaps are inserted inside structural regions of $G$. For example, an insertion occurs within a bracketed segment of the $G$ structure, splitting a stem arm in the aligned representation. A second long insertion appears near the end of the structure string. As a result, the displayed alignment does not preserve the intended correspondence between the stem arms and pseudoknotted arms visible in Figure 1. This is a nucleotide-level behavior relative to the stem-level objective considered in this paper: the alignment is syntactically valid, but it does not provide a useful mapping between stems.

**More permissive gap parameters.** We repeated the same comparison with lower gap costs, using `gapOpen=3` and `gapExtend=1`. This makes long gaps less costly and should, in principle, allow the nucleotide-level alignment to place larger insertions outside conserved structural blocks.

**Listing 6** LaRA output with `balance=INF`, `gapOpen=3`, and `gapExtend=1`.

```
F sequence:  AGU__AGGGGUACCAGCAAACUCGGCAUUUUAUAGAAGGGGACGAUGUGCCGAAGUGAUGUCGCGCG
F structure: .((__(....))).....((((((..((((....)))).[[[[.....))).))))..]]]].....
G sequence:  GAUCCGAUUGUUGCGGCCGAC___GUAUCAGAC_GA___CGGAAAUACGGCGGU_UUAU_UUAAGU_
G structure: ...(((.(((.........[___[[[[....)_))___)))..]]]]]...((_((..._..))))_
```

Lowering the gap penalties does not remove this behavior in the example. Instead, gaps are still placed inside bracket runs, and the resulting alignment continues to split regions that should be treated as coherent stems. This behavior reflects the difference between optimizing a nucleotide-level alignment objective and directly optimizing a map between stems.

**Projection of the coarse alignment.** For comparison, Listing 7 shows a nucleotide-level projection of the coarse alignment produced by our method. The method first computes a partial map between stems. Matched stems are then projected to nucleotide resolution by aligning corresponding stem arms as blocks. The vertical bars mark the boundaries of coarse stem blocks.

**Listing 7** Nucleotide-level projection of the stem-level alignment. Vertical bars mark coarse stem boundaries.

```
F sequence:  |AGUAGGGGUACCAGCAA|ACU_CGG|CAUUUUAUAGAAGGG|GACG_|AUGUG|CCGAAGU|GA|UGUC_|GCGCG__________|
F structure: |.(((....))).....|(((_(((|..((((....)))).|[[[[_|.....|))).)))|..|]]]]_|.....__________|
G sequence:  |GAU_____________|CCGAUUG|UUGCGGCCGA_____|CGUAU|CAGA_|CGA_CGG|AA|AUACG|GCGGUUUAUUUAAGU|
G structure: |..._____________|(((.(((|.........._____|[[[[[|...._|)))_)))|..|]]]]]|...((((....))))|
```

The projection is not intended to replace a full nucleotide-level sequence–structure aligner. Rather, it shows the type of information produced by the coarse algorithm: corresponding stems are identified first, and nucleotide-level columns are introduced only after the stem map has been fixed. This avoids placing gaps inside a stem arm when the downstream task is to compare stems as coherent units.

In other LaRA runs, we also observed a 5′-anchoring behavior, where an early nucleotide was aligned before a leading gap even though introducing the gap would better align the downstream stems. Such cases further motivate the use of stem-level alignment as a preprocessing or anchoring step when the primary biological question concerns structural correspondence rather than nucleotide identity.

## C Rfam Samples

In our results we evaluated our method on Rfam RNA structures, specifically the first 10 of each family. Table 2 contains the accession numbers for the used samples.

**Table 2** Rfam family sample accession numbers.

| RF00001 | RF00003 | RF00008 | RF00165 | RF00174 | RF00381 | RF00507 |
|---|---|---|---|---|---|---|
| X01556.1 | X06810.1 | D00685.1 | NC_022103.1 | AF010496.1 | BC056833.1 | NC_006577.2 |
| 3-118 | 261-421 | 1-46 | 27934-27996 | 39869-39653 | 241-299 | 13601-13681 |
| X55260.1 | X14417.1 | AJ247116.1 | NC_009657.1 | AF010496.1 | AF242520.1 | NC_003045.1 |
| 3-119 | 177-340 | 133-53 | 27966-28028 | 105318-105540 | 225-280 | 13342-13422 |
| M16174.1 | Z11883.1 | Y14700.1 | NC_028814.1 | AF010496.1 | AF258463.1 | NC_006213.1 |
| 3-119 | 3441-3603 | 133-53 | 27384-27446 | 116971-117193 | 230-288 | 13341-13421 |
| X55267.1 | X14419.1 | AJ247123.1 | NC_038294.1 | AF193754.1 | D10706.1 | NC_026011.1 |
| 3-119 | 178-338 | 132-52 | 29868-29928 | 4343-4143 | 256-314 | 13485-13565 |
| M16172.1 | X14416.1 | AJ247121.1 | NC_019843.3 | AF193754.1 | AC004152.1 | NC_004718.3 |
| 3-119 | 171-331 | 133-53 | 29875-29935 | 24966-24789 | 16641-16699 | 13399-13476 |
| AF001265.1 | X14415.1 | AJ247122.1 | NC_003436.1 | AF263012.1 | AF291576.1 | NC_022103.1 |
| 6033-6149 | 175-335 | 132-52 | 27820-27882 | 9229-9016 | 215-270 | 12533-12607 |
| X05057.1 | X14414.1 | AJ247113.1 | NC_038861.1 | AF306632.1 | D11373.1 | NC_002306.3 |
| 3-119 | 152-312 | 134-53 | 28363-28425 | 382-182 | 473-530 | 12410-12484 |
| X15126.1 | Z11883.1 | AJ536620.1 | NC_028806.1 | AJ000758.1 | BC054690.1 | NC_038861.1 |
| 3-120 | 2839-2998 | 1-40 | 27893-27955 | 1191-1369 | 204-259 | 12339-12413 |
| Z50737.1 | Z11883.1 | AJ536617.1 | NC_002306.3 | AJ000758.1 | AB017117.1 | NC_028806.1 |
| 3-119 | 1496-1656 | 1-40 | 29138-29200 | 1489-1667 | 228-286 | 12325-12399 |
| X55261.1 | Z11883.1 | Y12833.1 | NC_032107.1 | AJ278144.1 | AF196332.1 | NC_017083.1 |
| 3-119 | 361-521 | 339-285 | 28249-28311 | 22351-22164 | 69-127 | 13520-13600 |

# Constructing Incompatibility Graphs of Pairs of Trees in Optimal Output-Sensitive Time

**Manuel Lafond**
Université de Sherbrooke, Canada

## Abstract

We present an output-sensitive algorithm for constructing incompatibility graphs between pairs of rooted or unrooted phylogenetic trees, in which edges represent incompatible clusters or splits. Incompatibility graphs capture conflicting evolutionary signals and play an important role in applications such as phylogenetic network reconstruction, supertree inference, and BHV distance computation. Existing approaches typically require $O(n^3/w)$ time using bitset operations, where $n$ is the number of taxa and $w$ is the machine word size. We introduce a new algorithm based on lowest common ancestor mappings that constructs the incompatibility graph of two rooted trees in optimal $O(n + d)$ time, where $d$ is the number of incompatibility edges. The method is extended to unrooted trees and trees with different leaf sets while preserving the same complexity, while also being relatively simple to implement. Experimental results on random and simulated phylogenetic trees show substantial practical speedups over existing implementations, particularly on sparse incompatibility graphs, which commonly arise in large datasets.

**2012 ACM Subject Classification** Theory of computation → Design and analysis of algorithms; Applied computing → Computational biology

**Keywords and phrases** Phylogenetics, graph theory, output-sensitive algorithms, clusters, splits, incompatibility

**Digital Object Identifier** 10.4230/LIPIcs.WABI.2026.25

**Supplementary Material**
*Software (Code and data)*: `https://github.com/manuellafond/incompa-tree/`

**Funding** ML acknowledges financial support from the Natural Sciences and Engineering Research Council of Canada (NSERC).

## 1 Introduction

In phylogenetics, the study of cluster and split incompatibility offers important insight into how and why conflicting evolutionary signals arise. In this setting, clusters (subsets of taxa forming clades in a rooted tree) and splits (bipartitions of taxa separated by an edge in an unrooted tree) offer complementary ways to represent phylogenies, and incompatibilities among them reflect discordance between inferred relationships. Such contradictory signals across different data sources or inference methods are often not merely noise, but can point to complex evolutionary processes. Indeed, they have important applications ranging from improving the accuracy of phylogenetic tree reconstruction [24, 9, 10] to identifying events such as horizontal gene transfer, hybridization, and recombination [14, 26, 20].

In many situations, clusters of interest come from two or more (rooted) trees, which are are possibly built from different tools or represent different evolutionary hypotheses. The incompatibility graph of two trees $T, T'$ has a vertex for every internal node of $T$ and $T'$, and two vertices share an edge if their corresponding nodes have incompatible clusters, i.e., no tree can contain both clusters simultaneously. In unrooted trees, the same type of graph exists, but with clusters being replaced by splits. In this work, our aim is to reconstruct incompatibility graphs of pairs of rooted or unrooted trees as fast as possible.


© Manuel Lafond;
licensed under Creative Commons License CC-BY 4.0

Incompatibility graphs have several applications in phylogenetics. They are notably used by the CASS algorithm [26], which aims to reconstruct a phylogenetic network of small level that displays a given set of clusters. The incompatibility graph is an important component of this approach, since its connected components can be handled separately and, importantly, correspond to the biconnected components (blobs) of the output network. While the approach can deal with arbitrary sets of clusters in a heuristic manner, the CASS algorithm was shown to correctly minimize the level of the output network if it is given the set of clusters of two trees [17]. Therefore, faster incompatibility graph algorithms for this case can be useful for network reconstruction.

Another important motivation of incompatibility graphs is their central role in the Owen-Provan algorithm [23], a celebrated polynomial-time algorithm to compute the *BHV distance* between edge-weighted unrooted trees [3]. This distance projects such trees in a space where dimensions correspond to splits and asks for a shortest geodesic path in this space. The algorithm starts with the incompatibility graph $G$ of splits of the two trees and computes up to $n$ vertex covers on $G$, which determine successive directions of the desired geodesic path (this is *very* roughly speaking, see [23] for details). The current implementation [22] builds $G$ by comparing all pairs of nodes while using bitsets for set operations, which altogether takes time $O(n^3/w)$ ($n$ is the number of leaves and $w$ is the word size). Each subsequent vertex cover computation takes time $O(n^3)$. Although the graph construction is not the bottleneck in theory, in practice it can sometimes dominate the running time (in our experiments, we saw that it does when the trees share many splits). Moreover, the vertex cover computations could be accelerated by recent improved flow techniques [8], and the incompatibility graph construction may eventually become both the theoretical and practical bottleneck.

Let us also note that a maximum independent set of the incompatibility graph yields a set of splits or clusters that fit into a tree. This corresponds to a notion of supertree sometimes known as a Maximum Split Fit [24, 9]. Although finding such a supertree is NP-hard [1], some heuristics do make use of the incompatibility graph (see e.g. [10]). On a similar note, we remark that *loose consensus trees*, which contain all the clusters from a set of trees that have no incompatibility at all [5], can be built directly from the incompatibility graphs, as it suffices to combine clusters corresponding to isolated vertices. There already exists an optimal linear-time algorithm to build loose consensus trees [16], but fast incompatibility graph construction algorithms can nonetheless provide alternative approaches.

In this work, our aim is to accelerate the construction of the incompatibility graph between two trees $T$ and $T'$, which is used by the aforementioned methods. We are not aware of any algorithm that achieves a better time than $O(n^3/w)$ as mentioned above, although $O(n^2)$ seems doable with appropriate data structures. On the other hand, it would appear that any algorithm needs to run in time $\Omega(n^2)$, because the incompatibility graph may have a quadratic number of edges and we need to write those edges to the output.

A more refined lower bound can be established though. If the graph has $d$ edges, then the total size of the output graph is $O(n + d)$, which is a more accurate lower bound to aim for. An algorithm whose running time is dependent on the size of the input *and* the size of the output is called an *output-sensitive algorithm*. A recent example of this type of algorithm was developed by Weller [27], who showed that one can enumerate all conflicting rooted triples between a pair of trees in optimal time $O(n + t)$, with $t$ the number of conflicting triples. In our work, we show that incompatiblity graphs can indeed be computed in optimal output-sensitive time $O(n+d)$. Note that the number of edges $d$ of the output is not given as input, so the algorithm runs in optimal time while being oblivious to the lower bound. The algorithm works on rooted trees by applying various techniques based on the *lca-mapping* borrowed from gene tree and species tree reconciliation [4, 11, 6, 18]. We then show how this algorithm can be used on unrooted trees with the same running time.

While the complexity and correctness analysis of the algorithm are not trivial, it is not difficult to implement. We evaluated the efficiency gains of our C++ implementation against the basic $O(n^3/w)$ time algorithm. For the latter, we used our own implementation as well as the one provided by the authors of the Owen-Provan algorithm. All methods were first tested on random trees, and as we show, the density of the incompatibility graphs of random trees decreases rapidly as the number of leaves grows. This means that the number of edges $d$ gets closer to linear as $n$ grows, making the $O(n+d)$ complexity especially relevant in this case – as demonstrated by our empirical gains of efficiency of several orders of magnitude. We also evaluate trees that, instead of being completely random, are more likely to occur in evolution, by means of a comparison on simulated gene trees using SimPhy [21]. We studied varying levels of incomplete lineage sorting, which we show are correlated with the density of incompatibility graphs. Again, the results demonstrate that significant gains in time can be obtained with our new algorithm.

The implementation of the algorithm and all data necessary to reproduce the experiments are available at `https://github.com/manuellafond/incompa-tree/`.

## 2 Preliminary notions

We begin by focusing on rooted trees. We thus use the term *tree* to refer to a rooted tree. We deal with unrooted trees in a much later section, where we shall introduce the additional necessary definitions. In this setting, a tree $T$ is a directed acyclic graph that has a single node $\rho(T)$ of indegree 0 called the *root*, such that every vertex other than the root has exactly one in-neighbor, called its *parent*. We also assume no node of $T$ has exactly one child (in particular, there is no node of indegree 1 and outdegree 1), and we do not assume binary trees. The parent of a node $v \in V(T)$ is denoted $parent_T(v)$, and we put $parent_T(\rho(T)) = \perp$, which we interpret as undefined. We write $V(T)$ for the set of nodes of $T$. A *leaf* of $T$ is a node of outdegree 0. We denote by $L(T)$ the set of leaves of $T$. We also write $I(T) = V(T) \setminus L(T)$, which is the set of *internal nodes* of $T$. Note, $\rho(T)$ is included in $I(T)$, unless $T$ has only one node (in that case, the root counts as a leaf).

The out-neighbors of a node $v$ of $T$ are called the *children* of $v$ and are denoted $ch_T(v)$. For $u, v \in V(T)$, we write $u \preceq_T v$ if $u$ is a *descendant* of $v$, i.e., there is a directed path from $v$ to $u$ in $T$. We call $v$ an ancestor of $u$, noting that $v$ is a descendant and an ancestor of itself. We may write $u \prec_T v$ if $u \neq v$ also holds, and call $u$ a *strict descendant* of $v$. Two nodes $u, v$ are *incomparable* if $u$ is not a descendant nor an ancestor of $v$.

The *cluster* of a node $u \in V(T)$, denoted $C_T(u)$, is the set of all leaves that descend from $u$. That is, $C_T(u) = \{\ell \in L(T) : \ell \preceq_T u\}$. We also denote $C(T) = \{C_T(u) : u \in V(T)\}$ for the set of clusters of $T$. Observe that if $v \in I(T)$, then $|C_T(v)| \geq 2$ because $v$ has at least two children. Figure 1 illustrates two trees whose internal nodes are labeled by their cluster.

Let $Z \subseteq V(T)$. We define $lca_T(Z)$ as the lowest common ancestor of $Z$, that is, the node $u$ of $T$ that satisfies $z \preceq_T u$ for all $z \in Z$, such that $u$ is at maximum distance from the root of $T$ with this property. If $|Z| = 2$, we may write $lca_T(x, y)$ instead of $lca_T(\{x, y\})$, where $\{x, y\} = Z$. When we mention the notion of an *lca query*, we mean that we would like to obtain $lca_T(x, y)$ for two given nodes $x, y$.

**Incompatibility graphs.** Two sets $A, B$ are *compatible* if there exists a tree $T$ such that $A$ and $B$ are both in $C(T)$. Otherwise, $A, B$ are *incompatible*. Note, a set with a single element is compatible with any set. Let $\mathcal{C}$ be a collection of sets. We say that $\mathcal{C}$ is *compatible* if there is a tree in which each element of $\mathcal{C}$ is a cluster. The following results are well-known.

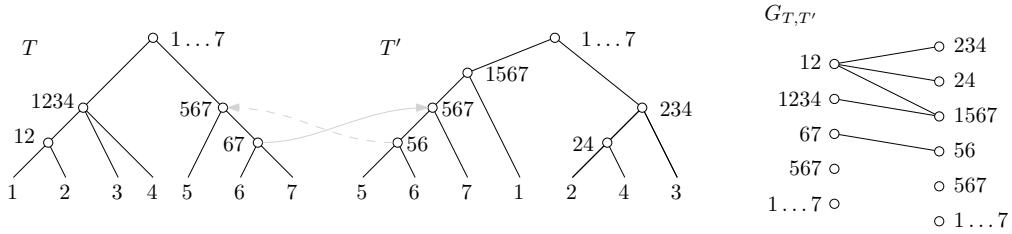



**Figure 1** Two trees $T, T'$ with $L(T) = L(T') = \{1, 2, 3, 4, 5, 6, 7\}$ and their incompatibility graph $G_{T,T'}$. Each internal node $u$ of $T$ (resp. $x$ of $T'$) is labeled with $C_T(u)$ (resp. $C_{T'}(x)$). The vertices of $G_{T,T'}$ are labeled with their corresponding cluster. The light gray arrows between $T$ and $T'$ represent the lca-map of two nodes. The node $u$ of $T$ whose cluster is $\{6, 7\}$ has $\vec{\mu}(u) = lca_{T'}(\{6, 7\})$ and the node $x$ of $T'$ whose cluster is $\{5, 6\}$ has $\overleftarrow{\nu}(x) = lca_T(\{5, 6\})$.

▶ **Proposition 1** ([25, 13]). *Two sets $A, B$ are compatible if and only if at least one of $A \cap B = \emptyset, A \subseteq B$, or $B \subseteq A$ is true.*

*Moreover, a collection of sets $\mathcal{C}$ is compatible if and only if every two elements $A, B \in \mathcal{C}$ are compatible. Moreover, if $\mathcal{C}$ is compatible, there exists a tree $T$ satisfying $C(T) = \mathcal{C}$.*

Let $T, T'$ be two trees satisfying $I(T) \cap I(T') = \emptyset$. Two nodes $u \in I(T), x \in I(T')$ are *incompatible* if $C_T(u)$ and $C_{T'}(x)$ are incompatible. The *incompatibility graph* of $T, T'$, denoted $G_{T,T'}$, is the undirected bipartite graph with vertex set $V(G_{T,T'}) = I(T) \cup I(T')$ and edge set

$$E(G_{T,T'}) = \{ux : u \in I(T), x \in I(T'), u \text{ and } x \text{ are incompatible}\}.$$

See Figure 1 for an illustration. For example, the node of $T$ with cluster $\{1, 2, 3, 4\}$ is incompatible with the node of $T'$ with cluster $\{1, 5, 6, 7\}$ because the two sets intersect and none is a subset of the other. On the other hand, $\{1, 2, 3, 4\}$ is compatible with every other cluster of $T'$ because it either has no intersection with them, contains them, or is contained by them.

Let $n = \max(|L(T)|, |L(T')|)$. There is a straightforward $O(n^3)$ time algorithm to compute $G_{T,T'}$. First, compute $C_T(u)$ for each $u \in V(T)$, which can easily be done in total time $O(n^2)$. The clusters $C_{T'}(x)$ of $T'$ can be computed similarly. Then for each $u \in V(T)$ and each $x \in V(T')$, check whether one of $C_T(u) \cap C_{T'}(x) = \emptyset, C_T(u) \subseteq C_{T'}(x)$, or $C_{T'}(x) \subseteq C_T(u)$ holds, and if not add $\{u, x\}$ to $E(G_{T,T'})$. The latter check takes time $O(n)$ and there are $O(n^2)$ pairs to check, which results in time $O(n^3)$. If the clusters are represented as bit vectors on $n/w$ words each of size $w$, then the last check can be accelerated to time $O(n/w)$, for a total of $O(n^3/w)$. We aim for a running time of $O(|V(G_{T,T'})| + |E(G_{T,T'})|)$.

To finish this section, our algorithm will make use of the famous $O(n)$ time pre-processing algorithm that allows answering lca queries in constant time.

▶ **Theorem 2** ([15, 2]). *Let $T$ be a tree. Then there exists a data structure that, after a pre-processing time of $O(|L(T)|)$ on input $T$, can answer any lca query on $T$ in time $O(1)$.*

## 3 Computing incompatibility graphs

We assume for now that we are given two (rooted) trees $T$ and $T'$ with the same set of leaves $L(T) = L(T')$. We will show how to deal with unrooted trees in Section 3.3, and how to deal with trees having different leaf sets in Section 3.4.

We denote $n = |L(T)| = |L(T')|$. We also let $d = |E(G_{T,T'})|$ denote the number of edges in the incompatibility graph. Recall that $d$ is not given and is thus not known in advance – our algorithm needs to run in time $O(n + d)$ without prior knowledge of $d$.

## 3.1 Incompatibility through lca-maps

We now introduce the notion of lca-maps to characterize incompatible nodes.

- The *lca-map from $T$ to $T'$* is the map $\vec{\mu} : V(T) \to V(T')$ satisfying $\vec{\mu}(u) = lca_{T'}(C_T(u))$ for all $u \in V(T)$.
- The *lca-map from $T'$ to $T$* is the map $\overleftarrow{\nu} : V(T') \to V(T)$ satisfying $\overleftarrow{\nu}(x) = lca_T(C_{T'}(x))$ for all $x \in V(T')$.

In Figure 1, the gray arrow from $T$ to $T'$ illustrates $\vec{\mu}(u)$, where $C_T(u) = \{6, 7\}$, and the dashed arrow from $T'$ to $T$ illustrates $\overleftarrow{\nu}(x)$, where $C_{T'}(x) = \{5, 6\}$.

Note that since we assume $L(T) = L(T')$, $\vec{\mu}$ and $\overleftarrow{\nu}$ are well-defined. Notation-wise, we use arrows above the map names to help remember their "direction", i.e., if we imagine $T$ to the left of $T'$, $\vec{\mu}$ maps nodes from the left tree $T$ to the right tree $T'$, and $\overleftarrow{\nu}$ maps nodes the other way around. We assume that $\vec{\mu}$ and $\overleftarrow{\nu}$ are stored in a lookup table, which can be achieved for instance by assigning a unique identifier to each node in $V(T) \cup V(T')$ between 1 and $|V(T) \cup V(T')|$. This allows accessing each value $\vec{\mu}(u)$ or $\overleftarrow{\nu}(x)$ in constant time, once the table is constructed.

Note that $\vec{\mu}$ is monotone, in the sense that $u \preceq_T v$ implies $\vec{\mu}(u) \preceq_{T'} \vec{\mu}(v)$. Likewise, $\overleftarrow{\nu}$ is monotone. We will also use the following observation.

▶ **Observation 3.** *For any $u \in V(T)$, $C_T(u) \subseteq C_{T'}(\vec{\mu}(u))$. Moreover, $C_T(u)$ is not a subset of $C_{T'}(y)$ for any child $y \in ch_{T'}(\vec{\mu}(u))$.*

*Likewise, for any $x \in V(T')$, $C_{T'}(x) \subseteq C_T(\overleftarrow{\nu}(x))$. Moreover, $C_{T'}(x)$ is not a subset of $C_T(w)$ for any child $w \in ch_T(\overleftarrow{\nu}(x))$.*

**Proof.** For $u \in V(T)$, $C_T(u) \subseteq C_{T'}(\vec{\mu}(u))$ because by the definition of $\vec{\mu}$, $\vec{\mu}(u)$ has every element of $C_T(u)$ in its descendants. Let $y \in ch_{T'}(\vec{\mu}(u))$. Then $y$ cannot be an ancestor of every element of $C_T(u)$, since otherwise $y$ would be a lower common ancestor of $C_T(u)$ than $\vec{\mu}(u)$. Thus $C_T(u)$ cannot be a subset of $C_{T'}(y)$. The second statement of the observation about $x \in V(T')$ can be argued in the same manner. ◀

It is well-known from the area of gene tree and species tree reconciliation that the lca-maps can be computed in linear time, see for example [7, 28].

The lca-maps lead us to a useful characterization of node incompatibility. The conditions of the next lemma are illustrated in Figure 1, where the light gray arrows explain why $\{6, 7\}$ and $\{5, 6\}$ are incompatible.

▶ **Lemma 4.** *Two nodes $v \in I(T)$ and $x \in I(T')$ are incompatible if and only if all the following hold:*

**(1)** $C_T(v) \cap C_{T'}(x) \neq \emptyset$*;*

**(2)** $x \prec_{T'} \vec{\mu}(v)$*;*

**(3)** $v \prec_T \overleftarrow{\nu}(x)$*.*

**Proof.** Let us start with the forward direction. If $v, x$ are incompatible, this means that $C_T(v)$ and $C_{T'}(x)$ are incompatible sets. In that case, condition (1) $C_T(v) \cap C_{T'}(x) \neq \emptyset$ must hold by Proposition 1. For (2), first note that we have $C_T(v) \subseteq C_{T'}(\vec{\mu}(v))$ by Observation 3. Let $z \in C_T(v) \cap C_{T'}(x)$ (such a $z$ exists by (1)). Then $z$ is in both the cluster of $x$ and the

cluster of $\vec{\mu}(v)$. Thus both $x$ and $\vec{\mu}(v)$ are ancestors of $z$. This implies that $x$ and $\vec{\mu}(u)$ cannot be incomparable. Now suppose that $x$ is not a strict descendant of $\vec{\mu}(v)$. Then $x$ must be an ancestor of $\vec{\mu}(v)$, with equality possible. This implies $C_T(v) \subseteq C_{T'}(\vec{\mu}(v)) \subseteq C_{T'}(x)$, which is a contradiction since it would mean that $v$ and $x$ are compatible by Proposition 1. Thus $x$ can only be a strict descendant of $\vec{\mu}(v)$. For (3), a symmetric argument shows that $v$ must be a strict descendant of $\overleftarrow{\nu}(x)$. Thus all required properties hold.

Conversely, suppose that $v$ and $x$ satisfy conditions 1-2-3 of the statement. We show that none of the compatibility conditions of Proposition 1 is true. We already know by condition (1) of the lemma that $C_T(v) \cap C_{T'}(x) \neq \emptyset$. Moreover, because $|C_T(v)| \geq 2$ and because $\vec{\mu}(v)$ is the lowest common ancestor of $C_T(v)$, $\vec{\mu}(v)$ has at least two distinct children $y_1, y_2$ that are ancestors of at least one leaf in $C_T(v)$ (if $\vec{\mu}(v)$ had only one child that is an ancestor of all $C_T(v)$, this would contradict Observation 3). Since (2) holds, $x$ is a strict descendant of $\vec{\mu}(v)$, which means that one such child, say $y_1$, does not have $x$ as a descendant nor an ancestor, and thus $C_{T'}(x) \cap C_{T'}(y_1) = \emptyset$. Therefore, $C_T(v) \setminus C_{T'}(x)$ contains some element of $C_T(v) \cap C_{T'}(y_1) \neq \emptyset$, and we deduce that $C_T(v)$ is not a subset of $C_{T'}(x)$. By the same reasoning, condition (3) stating that $v \prec_T \overleftarrow{\nu}(x)$ implies that $\overleftarrow{\nu}(x)$ has a child $w_1$ that is incomparable with $v$, such that $C_T(w_1)$ contains an element of $C_{T'}(x)$ that is not in $C_T(v)$. Thus $C_{T'}(x)$ is not a subset of $C_T(v)$. None of the conditions of Proposition 1 hold, and thus $v$ and $x$ are not compatible. ◀

The previous lemma suggests an algorithm to build $G_{T,T'}$. For each $v \in I(T)$, we can start a traversal of $T'$ at $\vec{\mu}(v)$ and explore its strict descendants $x$ that satisfy $v \prec_T \overleftarrow{\nu}(x)$ and $C_T(v) \cap C_{T'}(x) \neq \emptyset$. All such nodes satisfy the conditions of Lemma 4, and the time spent to check the conditions can be "charged" to an edge inserted into $G_{T,T'}$. When we reach a node $x$ such that $\overleftarrow{\nu}(x)$ is not a strict ancestor of $v$, or such that $C_T(v) \cap C_{T'}(x) = \emptyset$, we do not explore the descendants of $x$ further. This would work correctly, but unfortunately there are two problems with this idea. First, we are not aware of an efficient pre-processing to determine quickly whether $C_T(v) \cap C_{T'}(x) = \emptyset$. Moreover, even if such a pre-processing existed, stopping when we reach a node compatible with $v$ implies that we waste time making compatibility checks that do not lead to edges in $G_{T,T'}$. This wasted time thus cannot be "charged" to the edges of $G_{T,T'}$, and the number of wasted compatibility checks could be arbitrarily larger than $d$, which is an obstacle to our desired output-sensitive running time.

Nonetheless, the characterization Lemma 4 is still useful, as it is the basis for the following alternate characterization, which sidesteps the need for set containment and intersection computations. The lemma identifies two conditions for nodes $v \in V(T)$ and $x \in V(T')$ to be incompatible, which are illustrated in Figure 2. See the caption therein for an intuition of the necessity of having at least one of those conditions. It turns out that these two are also the *only* possible reasons for incompatibility.

▶ **Lemma 5.** *Two nodes $v \in I(T)$ and $x \in I(T')$ are incompatible if and only if $v \prec_T \overleftarrow{\nu}(x)$, and in addition at least one of the following is true:*

**(a)** *$v$ has a child $w$ such that $\vec{\mu}(w) \preceq_{T'} x \prec_{T'} \vec{\mu}(v)$; or*

**(b)** *$v$ has a child $w$ such that $w$ and $x$ are incompatible.*

**Proof.** For the forward direction, suppose that $v, x$ are incompatible. Then conditions 1-2-3 of Lemma 4 are true. In particular, Lemma 4.(3) holds and therefore $v \prec_T \overleftarrow{\nu}(x)$ holds. We must next show that at least one (a) or (b) is true. If condition (a) holds, we are done, so let us assume this is not the case. Therefore, we assume that $v$ has no child $w$ such that $\vec{\mu}(w) \preceq_{T'} x \prec_{T'} \vec{\mu}(v)$. In other words, for any child $w$ of $v$, either $\vec{\mu}(w) \preceq_{T'} x$ is false, or $x \prec_{T'} \vec{\mu}(v)$ is false. By Lemma 4.(2) and the incompatibility of $v, x$, we know that $x \prec_{T'} \vec{\mu}(v)$ is true. Therefore, $\vec{\mu}(w) \preceq_{T'} x$ is false for every child $w$ of $v$.

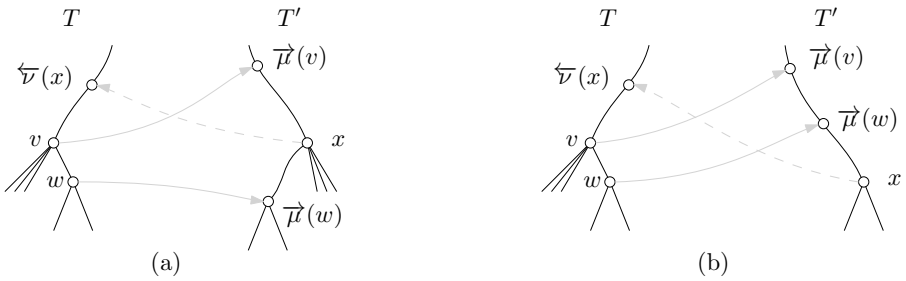



**Figure 2** The two situations that make $v, x$ incompatible in Lemma 5. Both situations require that $v \prec_T \overleftarrow{\nu}(x)$. (a) If $v$ has a child $w$ with $\overrightarrow{\mu}(w) \preceq_{T'} x \prec_{T'} \overrightarrow{\mu}(v)$, then $C_{T'}(x)$ contains $C_T(w)$ and therefore intersects with $C_T(v)$, and in addition $x \prec_{T'} \overrightarrow{\mu}(v)$ holds, so we have all conditions of Lemma 4. (b) If $v$ has a child $w$ that is incompatible with $x$, then Lemma 4 applies to $w$ and $x$. Thus $C_{T'}(x)$ intersects $C_T(w)$ and therefore intersects $C_T(v)$, and we have $x \prec_{T'} \overrightarrow{\mu}(w) \preceq_{T'} \overrightarrow{\mu}(v)$, and again we invoke Lemma 4 to deduce that $v, x$ are incompatible.

We want to show that condition (b) holds. By Lemma 4.(1), there exists $z \in C_T(v) \cap C_{T'}(x)$ since the latter is non-empty. Let $w$ be the child of $v$ such that $z \preceq_T w$ (this $w$ must exist). We argue that conditions 1-2-3 of Lemma 4 hold on $w$ and $x$. Note that $z \in C_T(w) \cap C_{T'}(x)$, so the intersection is non-empty and Lemma 4.(1) holds. For condition (2), in $T'$, both $\overrightarrow{\mu}(w)$ and $x$ are ancestors of $z$, so $\overrightarrow{\mu}(w)$ and $x$ cannot be incomparable. At the same time, we are assuming that $\overrightarrow{\mu}(w) \preceq_{T'} x$ is false. It follows that $x \prec_{T'} \overrightarrow{\mu}(w)$. As for condition (3), we have $w \prec_T \overleftarrow{\nu}(x)$ because $w \prec_T v$ and $v \prec_T \overleftarrow{\nu}(x)$ (using Lemma 4.(3) on $v$ and $x$). All conditions of Lemma 4 have been gathered and we can deduce that $w$ and $x$ are incompatible, and thus condition (b) holds.

For the converse direction, suppose that $v \prec_T \overleftarrow{\nu}(x)$ and that one of (a) or (b) holds. In particular, note that Lemma 4.(3) is true. Assume that condition (a) holds, so that $\overrightarrow{\mu}(w) \preceq_{T'} x \prec_{T'} \overrightarrow{\mu}(v)$ for some $w \in ch_T(v)$. In particular, Lemma 4.(2) is true. Moreover, $\overrightarrow{\mu}(w) \preceq_{T'} x$ and Observation 3 imply that $C_T(w) \subseteq C_{T'}(\overrightarrow{\mu}(w)) \subseteq C_{T'}(x)$. This implies that $C_{T'}(x) \cap C_T(v) \neq \emptyset$, which is the condition Lemma 4.(1). We have all the conditions of Lemma 4 and therefore $v, x$ are incompatible. So assume that that (b) holds, so that $x$ is incompatible with a child $w$ of $v$. By Lemma 4, $x \prec_{T'} \overrightarrow{\mu}(w)$ and thus $x \prec_{T'} \overrightarrow{\mu}(v)$ by the monotonicity of $\overrightarrow{\mu}$. Also by Lemma 4, $C_{T'}(x) \cap C_T(w) \neq \emptyset$ which implies $C_{T'}(x) \cap C_T(v) \neq \emptyset$. We already had $v \prec_T \overleftarrow{\nu}(x)$ and again, all the conditions of Lemma 4 have been gathered and so $v, x$ are incompatible. ◀

## 3.2 The algorithm

We need one last ingredient before proceeding. Let $x \in V(T')$. We define $diff_anc(x)$ as the closest ancestor $y$ of $x$ that satisfies $\overleftarrow{\nu}(y) \neq \overleftarrow{\nu}(x)$ (by closest, we mean at minimum distance from $x$). If no such ancestor exists, then $diff_anc(x) = \perp$, which is interpreted as undefined. The values of $diff_anc$ for every $x \in V(T')$ can easily be pre-processed in time $O(n)$ in a preorder traversal. That is, in a top-down fashion we can use the recurrence that puts $diff_anc(\rho(T')) = \perp$, and for a non-root node $x$ we have

$$diff_anc(x) = \begin{cases} diff_anc(parent_{T'}(x)) & \text{if } \overleftarrow{\nu}(x) = \overleftarrow{\nu}(parent_{T'}(x)), \\ parent_{T'}(x) & \text{otherwise.} \end{cases}$$

The pseudo-code of our procedure is given in Algorithm 1 (note, we use $G$ instead of $G_{T,T'}$ in the algorithm). At a high level, our algorithm uses Lemma 5 as follows. It iterates over each $v \in I(T)$, and for each such $v$ it checks for incompatibilities that could arise from

either condition (a) or (b). Condition (b) is the easiest to implement and is performed by the function *check_incomp_child*. In essence, condition (b) says that we can look at every $x$ incompatible with a child $w$ of $v$, and if $v \prec_T \overleftarrow{\nu}(x)$, then the node $x$ is automatically incompatible with $v$. The algorithm maintains an array $Q$ of visited nodes to prevent adding the same edge multiple times. That is, while iterating on $v$, we have $Q[x] = v$ if and only the edge $\{v, x\}$ was added (and the same $Q$ is re-used for all $v$'s).

**Algorithm 1** The main algorithm to build $G_{T,T'}$, denoted $G$ in the algorithm.

```
1  function build_incompatibility_graph(T, T')
2      Pre-process T, T' for O(1) lca queries
3      Construct the lca-maps μ⃗ and ν⃖
4      Pre-process T' for O(1) diff_anc(x) queries
5      Let G be a graph with vertex set V(T) ∪ V(T') and no edge
6      Let Q be a global array of length |V(T')|, with values initialized to −1
7      for each v ∈ I(T) in a postorder traversal do
8          check_between(v, G)
9          check_incomp_child(v, G)
10
11 function check_between(v, G)
12     for each child w of v do
13         x = μ⃗(w)
14         while x ≠ ⊥ and ν⃖(x) ⪯_T v do
15             x = diff_anc(x)
16         while x ≠ ⊥ and x ≺_T' μ⃗(v) and Q[x] ≠ v do
17             Add {v, x} to E(G)
18             Q[x] = v
19             x = parent_T'(x)
20
21 function check_incomp_child(v, G)
22     for each child w of v do
23         for each edge {w, x} incident to w in G do
24             if v ≺_T ν⃖(x) and Q[x] ≠ v then
25                 Add {v, x} to E(G)
26                 Q[x] = v
```

Condition (a) is implemented in the function *check_between*. The condition says that we also need to do a search from every $\vec{\mu}(w)$, and look for ancestors $x$ of $\vec{\mu}(w)$ satisfying $x \prec_{T'} \vec{\mu}(v)$ and $v \prec_T \overleftarrow{\nu}(x)$. We thus need a fast way to start from $\vec{\mu}(w)$ and "jump" to its first ancestor $x$ satisfying $v \prec_T \overleftarrow{\nu}(x)$, if any. Thanks to the fast $diff_anc$ queries and some arguing, the loop on line 14 makes a constant number of iterations and only take time $O(1)$. Once we have found this first ancestor $x$, we traverse all ancestors $x'$ of $x$ that also satisfy $x' \prec_{T'} \vec{\mu}(v)$ and stop when this becomes false. To visualize this, consider Figure 2.(a). If we imagine that $x$ in the subfigure is the first ancestor of $\vec{\mu}(w)$ satisfying $v \prec_T \overleftarrow{\nu}(x)$, then every node on the path from $x$ to $\vec{\mu}(v)$, excluding the latter, is incompatible with $v$. Therefore, the check for condition (a) can be seen as traversing a set of subpaths of $T'$ that contain incompatible nodes with $v$. We just need to ensure that we do not waste time going through the same subpaths multiple times, which is the role of the $Q$ array.

We can now state our main result.

▶ **Theorem 6.** *Algorithm 1 correctly computes the incompatibility graph $G_{T,T'}$ of two trees $T, T'$ with the same leaves in time $O(n+d)$, where $n$ is the number of leaves of the input trees, and $d$ is the number of edges of $G_{T,T'}$.*

**Proof sketch.** The full proof is a bit lengthy, so it can be found in the Appendix. Here we provide a sketch. The correctness essentially follows from Lemma 5. If $v, x$ satisfy condition (a) of the lemma, then *check_between* will catch it and add the edge $\{v, x\}$, and if it satisfies condition (b), then *check_incomp_children* will catch it. There are some subtleties though. In particular, it is important for correctness to perform *check_between* before *check_incomp_child*. This is because we would not want *check_incomp_child* to put some $Q[x] = v$, which could later prevent *check_between* to explore subpaths above that $x$ (see the correctness proof for details).

For the running time, we already know that every pre-processing step takes time $O(n)$. Consider the time spent in *check_between*. It loops through the children of each node $v \in V(T)$, and overall throughout the execution of the whole algorithm, this corresponds to going through every parent-child relation of $T$, of which there are $O(n)$. The **while** loop on line 14 starts at $x = \vec{\mu}(w)$, and then loops through ancestors until $v \prec_T \overleftarrow{\nu}(x)$ (or $x = \perp$). We can argue that the loop makes at most three iterations, because in the worst case we start with $\overleftarrow{\nu}(x) = w$, then after one $diff_anc$ we get $\overleftarrow{\nu}(x) = v$ or above, and then after one more $diff_anc$, $\overleftarrow{\nu}(x)$ if definitely a strict ancestor of $v$ (or $\perp$). Therefore the loop actually takes time $O(1)$ per $w$. As for the loop on line 16, every time we enter it, we add an edge to the graph $G$, and thus the total time spent in this loop throughout the whole execution is $O(d)$ (the array $Q$ of "visited" nodes is needed here to ensure we add each edge exactly once). Note, checking whether $v \prec_T \overleftarrow{\nu}(x)$ can be done in time $O(1)$ using lca queries. Hence *check_between* takes time $O(n+d)$. Then, *check_incomp_child* also enumerates the $O(n)$ parent-child relations, and the loop on line 23 amounts to enumerating every edge of $G$ throughout the whole execution, of which there are $O(d)$. The code inside that loop then takes constant time, so that subfunction also takes time $O(n+d)$. ◀

## 3.3 Extending to splits and unrooted trees

We now deal with unrooted trees, which have splits instead of clusters. The short version for the reader that is in a hurry: to obtain the incompatibility graph of splits of two unrooted trees on the same leaves, we root both trees at the same arbitrary taxon and run our algorithm (note, a one-sentence version of this idea appears in [12]). This seems to be a folklore procedure, but we could not find a proof of correctness of this way of doing, so we provide the details below.

An *unrooted tree* $U$ is an acyclic connected graph with no vertex of degree 2. We write $V(U)$ and $E(U)$ for the set of vertices and edges of $U$, respectively. We also write $L(U)$ for the set of leaves of $U$, i.e., its set of vertices that have a single neighbor. A *split* of a set $X$ is a partition $\{A, B\}$ of $X$, so that $A, B$ are non-empty and non-intersecting and $A \cup B = X$. We denote such a split by $A|B$, noting that $B = X \setminus A$ and $A = X \setminus B$. Given an unrooted tree $U$, removing an edge $uv \in E(U)$ results in two connected components, and the set of leaves present in these components results in a split $A|B$ of $L(U)$, which we call the *split of* $uv$ and denote by $S_U(uv)$. The set of splits of $U$ is $S(U) = \{S_U(uv) : uv \in E(U)\}$.

Two splits $A|B$ and $C|D$ of a set $X$ are *compatible splits* if there exists an unrooted tree $U$ with $L(U) = X$ that contains both splits. Given two unrooted trees $U, U'$, the *split incompatibility graph of* $U$ *and* $U'$ is the graph $G^{\mathsf{u}}_{U,U'}$ whose vertex set is $E(U) \cup E(U')$, with an edge $\{uv, xy\} \in E(G^{\mathsf{u}}_{U,U'})$ if and only if $S_U(uv)$ and $S_{U'}(xy)$ are incompatible splits (note, the $\mathsf{u}$ in the superscript of $G^{\mathsf{u}}_{U,U'}$ stands for "unrooted").

The following is well-known.

▶ **Proposition 7** ([25]). *Two splits $A|B, C|D$ of some set $X$ are compatible splits if and only if at least one of the intersections $A \cap C, A \cap D, B \cap C, B \cap D$ is empty.*

The next result shows how computing $G^{\mathsf{u}}_{U,U'}$ from unrooted trees reduces to computing $G_{T,T'}$ on rooted trees, for an appropriate rooting, with the same running time. See the Appendix for the proof.

▶ **Proposition 8.** *Let $U, U'$ be two unrooted trees with $L(U) = L(U') = \{1, 2, \ldots, n\}$ with $n \geq 3$, and let $T, T'$ be the rooted trees obtained from $U$ and $U'$, respectively, by assigning the neighbor of leaf 1 as the root in both trees.*

*Let $uv \in E(U)$ and $xy \in E(U')$, where $u$ is the parent of $v$ in $T$ and $x$ is the parent of $y$ in $T'$. Then $S_U(uv)$ and $S_{U'}(xy)$ are compatible splits if and only if $C_T(v)$ and $C_{T'}(y)$ are compatible clusters.*

## 3.4 Extending to trees with different leafsets

Going back to rooted trees, we now discuss how to deal with trees on different leaf sets. We show that this is mostly a small technicality, as we can add the missing leaves to the two trees without altering the incompatibility graph.

Suppose that $L(T) \neq L(T')$. If $L(T') \setminus L(T) \neq \emptyset$, define a tree $\tilde{T}$ obtained from $T$ as follows. Start from $T$, add a new root $r$ and new leaves $L(T') \setminus L(T)$, and add $\{\rho(T)\} \cup (L(T') \setminus L(T))$ as children of $r$. If $L(T') \setminus L(T) = \emptyset$, define $\tilde{T} = T$. Define $\tilde{T}'$ from $T'$ in a similar manner, i.e., add a new root to $T'$ whose children are $\rho(T')$ and $L(T) \setminus L(T')$ if the latter is non-empty, and otherwise define $\tilde{T}' = T'$. Note that if we create a new root in either tree, it has at least two children, so no internal node with a single child is created (proof in Appendix).

▶ **Proposition 9.** *Let $T, T'$ be two trees and let $\tilde{T}, \tilde{T}'$ be defined as above. Then $|V(G_{\tilde{T},\tilde{T}'})| \leq |V(G_{T,T'})| + 2$. Moreover, $ux$ is an edge of $G_{T,T'}$ if and only if $ux$ is an edge of $G_{\tilde{T},\tilde{T}'}$.*

We can easily construct $\tilde{T}$ and $\tilde{T}'$ from $T$ and $T'$ in time $O(n)$. It then suffices to construct $G_{\tilde{T},\tilde{T}'}$ and remove $\rho(\tilde{T})$ and $\rho(\tilde{T}')$ to obtain $G_{T,T'}$. Since the two incompatibility graphs have the same size (number of vertices plus edges) with the exception of two isolated vertices, the complexity of building $G_{\tilde{T},\tilde{T}'}$ is the same as that of $G_{T,T'}$.

## 3.5 Extending to more than two trees

As stated in the introduction, there are some approaches that use the incompability graph of possibly more than two trees. Let $\mathcal{T} = \{T_1, \ldots, T_k\}$ be a set of $k$ trees. The graph $G_{\mathcal{T}}$ has vertex set $\bigcup_{i=1}^{k} I(T_i)$ and edge set $\bigcup_{1 \leq i < j \leq k} E(G_{T_i,T_j})$. Let us denote by $n$ the maximum number of leaves in one of the input trees. The naive implementation goes through every pair of trees and adds the incompatibility edges between them, which takes time $O(k^2 n^3/w)$ to construct $G_{\mathcal{T}}$.

If we let $d_{ij} = E(G_{T_i,T_j})$ for each distinct $i, j \in \{1, 2, \ldots, k\}$, and denote by $d$ the number of edges in $G_{\mathcal{T}}$, we can iterate over every pair of trees $T_i, T_j$ and add the edges found by our approach. This takes time $O(\sum_{1 \leq i < j \leq k}(n + d_{ij})) = O(k^2 n + \sum_{1 \leq i < j \leq k} d_{ij}) = O(k^2 n + d)$, with $d$ the number of edges of $G_{\mathcal{T}}$.

While this is potentially much faster than the naive idea, this complexity does not coincide with the lower bound. Indeed, with $k$ trees the size of the input is $O(kn)$ and the size of the output is $O(kn + d)$. We do not know whether this lower bound can be achieved. This would be surprising, as there is not even enough time to perform any kind of linear pre-processing

between the pairs of trees, even though we may need to add edges between vertices from all distinct pairs of trees. Perhaps some clever ideas can be borrowed from $O(nk)$ time algorithms for tree consensus, which achieve optimal time $O(nk)$ by iterating over the trees one by one while maintaining a single tree that captures the necessary information from the previous trees. This appears extremely difficult, if not impossible, so we leave this as an open problem.

**Open problem.** Find an algorithm that construct the incompatibility graph of $k$ trees with $n$ leaves in time $O(nk + d)$, or determine that no such algorithm exists.

Note, if the pairs of input trees have $\Omega(n)$ incompatibility edges on average, then $d \in \Omega(k^2 n)$, in which case $O(k^2 n + d)$ is optimal. So, an $O(nk + d)$ time algorithm would need to have improved complexity specifically on instances where pairs of trees have $o(n)$ incompatibility edges, on average.

## 4 Experiments

The main purpose of our experiments is quantify empirically the efficiency gains of Algorithm 1. We tested two C++ implementations for the computation of incompatibility graphs. The first is the implementation of Algorithm 1, which we call the *fast* implementation. The second is the $O(n^3/w)$ time algorithm that compares each pair of clusters from the two trees, as described at the end of Section 2. For faster intersection and set containment tests, we stored clusters using the efficient EWAHBoolArray implementation from [19], which are compressed bit arrays. We call this the *naive* implementation[1]. Note, we did not implement the pre-processing for $O(1)$ lca queries, and instead use a basic $O(h)$ time algorithm for lca queries, where $h$ is the height of the tree. Profiling showed that these lca queries consist of less than 1% of the running time.

To limit possible biases that could arise from using only our own implementations, we also included the incompatibility graph computation used in the Owen-Provan algorithm [23] in our comparisons – we used the authors' implementation provided here: [22]. The latter implements the polynomial-time algorithm to compute the BHV distance between trees mentioned in the introduction. Computing the incompatibility graph is the first step, with the second step consisting of vertex cover computations. Since we are only interested in the first step, we made a minor modification to the code to calculate only the time needed to compute the incompatibility graph. As a small remark, the algorithm has a pre-processing step that calculates splits that are common to both trees. This can sometimes accelerate computation, since only splits that are on the same side of a common edge need to be compared for incompatibility. As we also count the pre-processing time in our implementation, we included the time to calculate the time to find common splits of the Owen-Provan algorithm[2]. The reader should bear in mind that the authors of the Owen-Provan algorithm did not aim to optimize the incompatibility graph step, and it is implemented in Java while our code is C++, so the comparisons should be taken with a grain of salt. All experiments were conducted on a standard laptop with 16 GB of RAM and a 3.2 GHz processor using a single CPU core.

[1] We also implemented a third version which represents clusters using a standard `std::set` object from C++. This was used to validate that all three methods produced the same incompatibility graph, on all pairs of trees (which was the case). We do not report the times for that third implementation, as it is about 10 times slower than the bit array version.

[2] To be fully specific, the timer starts at the beginning in the *getGeodesic* function, and stops right after the function *getIncidenceMatrix* has returned.

## 4.1 Datasets

We analyzed random trees and simulated gene trees with incomplete lineage sorting.

**Random trees.** For each $n \in \{20, 50, 100, 200, 500, 1000\}$, we generated 50 random binary trees with $n$ leaves. We used the standard randomization process where we start with a root containing all leaf labels, then choose a random partition into two non-empty sets, add two children to the root and pass them the respective labels sets, and then repeat recursively. In each tree, we then added a new leaf adjacent to the root, to simulate the reduction from unrooted to rooted trees described in Proposition 8.

With each implementation and each $n$, we computed the incompatibility graph for each of the $\binom{50}{2} = 1225$ pairs of trees that we generated.

**Simulated trees.** In order to analyze trees that are more likely to occur in evolution, we generated gene trees with SimPhy [21] under varying levels of incomplete lineage sorting (ILS). We turned off gene duplication, loss, and transfer events. In a nutshell, using SimPhy as such generates a species tree and then gene trees that evolve inside of it. With no ILS, all gene trees are identical to the species tree. As the amount of ILS increases, the gene trees maintain alleles for longer periods of time in the species branches, making them become increasingly different from the species tree, and from each other. The level of ILS therefore has an impact on the number of incompatibilities created between the gene trees. In SimPhy, the *population* parameter controls the amount of ILS, and we observed incompatibilities only at populations from $10^6$ and onwards.

So, for each $n \in \{20, 50, 100, 200, 500, 1000\}$ and each population $p \in \{10^6, 10^7, 10^8, 10^9\}$, we simulated 50 gene trees with SimPhy. As before, with each implementation and each $n$, we computed the incompatibility graph for each of the $\binom{50}{2} = 1225$ pairs of trees.

## 4.2 Results on random trees

The total running time to compare all pairs of random trees is shown in Figure 3. The x-axis is the number of leaves and the y-axis is the number of seconds needed to compare all pairs of random trees.

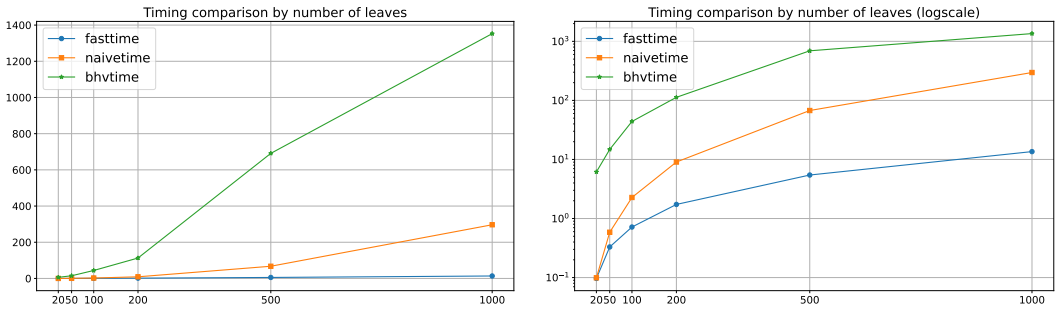



**Figure 3** Total running time to compare all $\binom{50}{2}$ pairs of random trees, per number of leaves. **x-axis**: number of leaves. **y-axis**: total time in seconds. On the left, a standard scale is used for the y-axis. On the right, the same data but using a logarithmic scale for the y-axis.

It is not surprising to see that the *fast* implementation is much faster, especially as the trees get larger. The magnitude of the speedup of our algorithm is easier to see using a logarithmic scale (on the right). The *fast* algorithm tends to be about 10 to 30 times

times faster than our *naive* implementation, and around 100 times faster than the BHV implementation. Notably, with the most interesting case of $n = 1000$, the times to compare all pairs of trees for $fast, naive$ and $bhv$ were respectively $13, 296$, and 1352 seconds.

**Density analysis: large trees have sparse incompatibility graphs.** We define the density of an incompatibility graph $G_{T,T'}$ as $\frac{|E(G_{T,T'})|}{|I(T)|\cdot|I(T')|}$. This is the number of edges of $G_{T,T'}$ divided by the maximum possible number of edges. Figure 4 illustrates the average density of the random trees as a function of $n$.

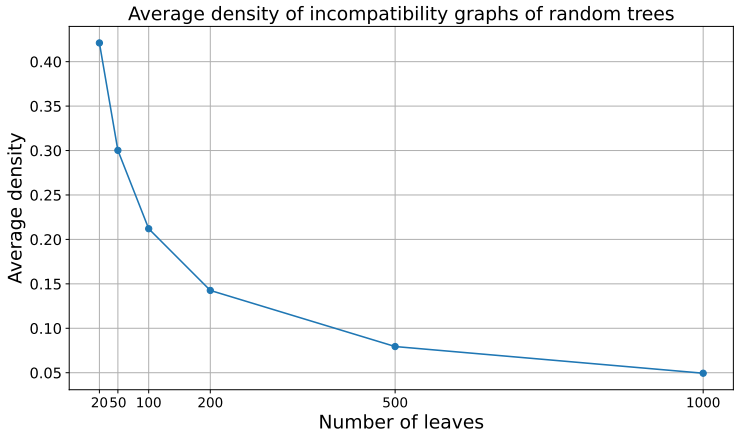



**Figure 4** Average density $G_{T,T'}$ for the pairs of random trees, with respect to the number of leaves.

The figure shows that the proportion of incompatibilities decreases quickly as we increase the size of the trees. We are not aware of any study that establishes the probability of two clusters from random trees to be incompatible, but there is at least an intuition that explains this. In a binary tree, "most" clusters are expected to be very small, in the sense that clusters of small constant size represent a large proportion of the clusters. As $n$ gets larger, such constant-size clusters are more likely to be non-intersecting with the clusters from the other tree. In other words, small clusters dominate the proportion of clusters, and the proportion of clusters they are incompatible with goes down with $n$.

One way to see this more concretely is to think incompatibilities between cherries in the trees, i.e., nodes whose children are both leaves, and therefore whose cluster has size 2. For example, if $T, T'$ are fully balanced binary trees, roughly half of the internal nodes in either tree are cherries. The cluster $\{x, y\}$ of a cherry of $T$ intersects only at most two cherries of $T'$, so the *proportion* of cherries of $T'$ that $\{x, y\}$ is incompatible with vanishes very quickly as $n$ grows. Then, if we consider clusters of size at most 4, they make up about 75% of the clusters in either tree, and using the same argument, the proportion of possible incompatibilities with subtrees of size 4 decreases as $n$ grows, and so on.

Characterizing the expected density of $G_{T,T'}$ of two random trees $T, T'$ as a function of $n$ is an interesting side question, but it is outside the scope of this paper. Nonetheless, we would argue that the fact that the density goes down with $n$ is a strong argument in favor of our algorithm. It means $d$ has a lesser impact in the $O(n + d)$ complexity when $n$ gets large.

## 4.3 Results on simulated gene trees

**Running time analysis.** Figure 5 shows the total running times on Simphy gene trees, where there is one curve for each combination of method and population. The tendencies are the same as in the random trees (notice, the curves are clustered by method for all population

sizes). We notice that the $fast$ implementation is slightly slower on larger populations. This is very likely because those produced denser graphs and therefore larger values of $d$, as shown in the next subsection.

In contrast, the $bhv$ implementation was noticeably faster on larger populations (the gap in the plot between population $10^9$ and the others is significant as it uses a logarithmic scale). This required investigating the code for an explanation. It turns out that the $bhv$ implementation is slower when the given trees are similar. This is implementation-specific and outside the scope of this paper, but in a nutshell, we found that this is caused by the pre-processing that enumerates common splits. If any such split exists, the algorithm repeats recursively on the matching subtrees on each side of the common split, thereby starting two new common split enumerations on the two smaller pairs of subtrees. So basically, this common splits enumeration happens as often as the number of common splits, making it slower on similar trees. This also explains why the total $bhv$ times on gene trees are higher than on the random trees, because random trees seem to rarely share the exact same splits.

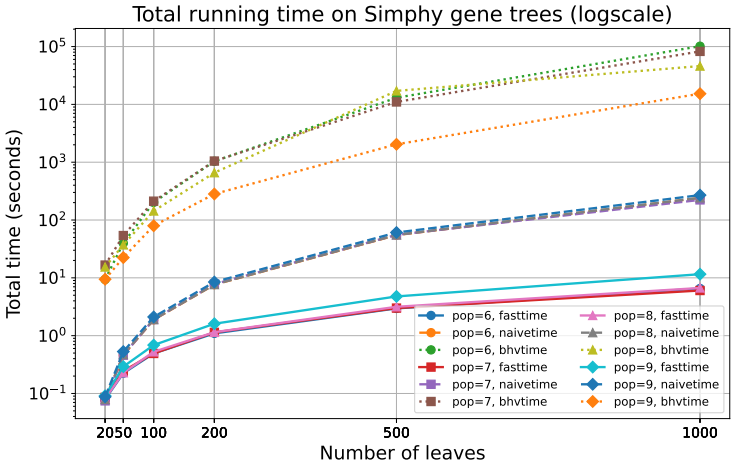



**Figure 5** Total running time to compare all $\binom{50}{2}$ pairs of gene trees, for each population, per number of leaves (in the legend, $pop = X$ means a population of $10^X$). **x-axis**: number of leaves. **y-axis**: total time in seconds. Full lines as $fast$, dashed lines are $naive$, and dotted lines are $bhv$. Note, the y-axis uses a logarithmic scale (the standard scale plot did not add anything meaningful, so we omit it).

**Density analysis.** Figure 6 shows the average density of the incompatibility graphs on the SimPhy datasets, for each number of leaves and population size. As in the random trees, we again observe a decrease in density as $n$ grows, highlighting once again the importance of output-sensitive algorithms for large $n$. Note, the curve for population size $pop = 1e6$ behaves a bit more erratically – this is probably because for the vast majority of pairs of trees $T, T'$, the graph $G_{T,T'}$ has 0 edges, so the few graphs that do have edges affect the average in more pronounced ways.

We also find it interesting that the amount of ILS in the trees has a direct impact on the proportion of incompatibilities. Although this is not surprising, it highlights that incompatibility graphs capture biologically meaningful disagreement between trees, rather than merely representing an abstract combinatorial object, which further motivates the need for efficient algorithms in downstream applications such as BHV distance computations and network inference.

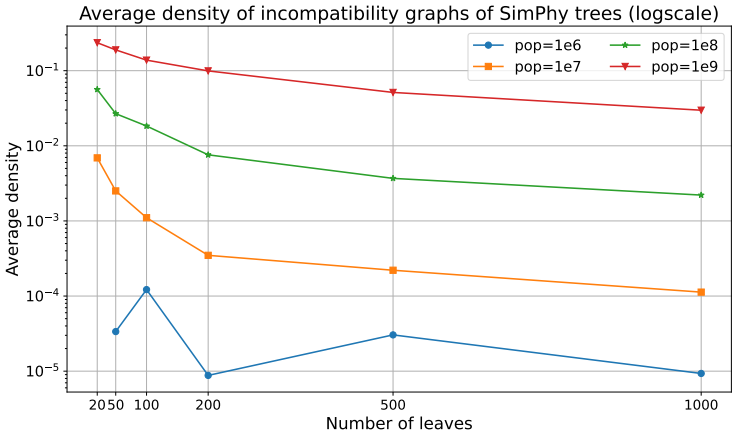



**Figure 6** Average density of $G_{T,T'}$ for the pairs of SimPhy gene trees, with respect to $n$ the number of leaves. There is one curve for each population size, from $1e6 = 10^6$ to $1e9 = 10^9$. For $pop = 1e6$ and $n = 20$, there is no point because the average density is 0, which cannot be displayed on a logarithmic scale.

## 5 Conclusion

In this work, we presented the first output-sensitive algorithm for constructing incompatibility graphs between pairs of rooted or unrooted phylogenetic trees. Beyond the theoretical improvement, our experiments demonstrate that incompatibility graphs arising from random and simulated data are often sparse in practice, making the output-sensitive complexity especially relevant for large datasets. In particular, we observed substantial empirical speedups over existing implementations used in BHV distance computations.

The algorithm opens several directions for future work. On the practical side, it would be interesting to integrate the method directly into software for BHV distance computation, phylogenetic network reconstruction, and supertree inference, in order to evaluate its impact on large-scale biological datasets. From a theoretical perspective, several questions remain open. Most notably, we do not know whether the incompatibility graph of $k$ trees can be constructed in optimal time $O(nk + d)$, where $d$ is the number of incompatibility edges. Another interesting direction is to better understand the expected structure and density of incompatibility graphs under probabilistic models of tree evolution, particularly in the presence of processes such as incomplete lineage sorting, hybridization, or horizontal gene transfer.

## References

1 Jonathan Badger, Paul Kearney, Ming Li, John Tsang, and Tao Jiang. Selecting the branches for an evolutionary tree.: A polynomial time approximation scheme. *Journal of Algorithms*, 51(1):1–14, 2004. doi:10.1016/S0196-6774(03)00086-5.

2 Michael A Bender and Martin Farach-Colton. The lca problem revisited. In *Latin American Symposium on Theoretical Informatics*, pages 88–94. Springer, 2000. doi:10.1007/10719839_9.

3 Louis J Billera, Susan P Holmes, and Karen Vogtmann. Geometry of the space of phylogenetic trees. *Advances in Applied Mathematics*, 27(4):733–767, 2001. doi:10.1006/AAMA.2001.0759.

4 Paola Bonizzoni, Gianluca Della Vedova, and Riccardo Dondi. Reconciling a gene tree to a species tree under the duplication cost model. *Theoretical computer science*, 347(1-2):36–53, 2005. doi:10.1016/J.TCS.2005.05.016.

5 Kåre Bremer. Combinable component consensus. *Cladistics*, 6(4):369–372, 1990.

6 Cedric Chauve and Nadia El-Mabrouk. New perspectives on gene family evolution: losses in reconciliation and a link with supertrees. In *Annual International Conference on Research in Computational Molecular Biology*, pages 46–58. Springer, 2009.

7 Kevin Chen, Dannie Durand, and Martin Farach-Colton. Notung: dating gene duplications using gene family trees. In *Proceedings of the fourth annual international conference on Computational molecular biology*, pages 96–106, 2000. doi:10.1145/332306.332351.

8 Li Chen, Rasmus Kyng, Yang Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. *Journal of the ACM*, 72(3):1–103, 2025. doi:10.1145/3728631.

9 Chris J Creevey and James O McInerney. Clann: investigating phylogenetic information through supertree analyses. *Bioinformatics*, 21(3):390–392, 2005. doi:10.1093/BIOINFORMATICS/BTI020.

10 Markus Fleischauer and Sebastian Böcker. Bad clade deletion supertrees: a fast and accurate supertree algorithm. *Molecular biology and evolution*, 34(9):2408–2421, 2017.

11 Pawel Górecki and Jerzy Tiuryn. Dls-trees: a model of evolutionary scenarios. *Theoretical computer science*, 359(1-3):378–399, 2006. doi:10.1016/J.TCS.2006.05.019.

12 Dan Gusfield. Efficient algorithms for inferring evolutionary trees. *Networks*, 21(1):19–28, 1991. doi:10.1002/NET.3230210104.

13 Dan Gusfield. *ReCombinatorics: the algorithmics of ancestral recombination graphs and explicit phylogenetic networks*. MIT press, 2014.

14 Dan Gusfield, Satish Eddhu, and Charles Langley. Optimal, efficient reconstruction of phylogenetic networks with constrained recombination. *Journal of bioinformatics and computational biology*, 2(01):173–213, 2004. doi:10.1142/S0219720004000521.

15 Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *siam Journal on Computing*, 13(2):338–355, 1984. doi:10.1137/0213024.

16 Jesper Jansson, Chuanqi Shen, and Wing-Kin Sung. Improved algorithms for constructing consensus trees. *Journal of the ACM (JACM)*, 63(3):1–24, 2016. doi:10.1145/2925985.

17 Steven Kelk, Celine Scornavacca, and Leo Van Iersel. On the elusiveness of clusters. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 9(2):517–534, 2011. doi:10.1109/TCBB.2011.128.

18 Manuel Lafond, Krister M Swenson, and Nadia El-Mabrouk. An optimal reconciliation algorithm for gene trees with polytomies. In *International Workshop on Algorithms in Bioinformatics*, pages 106–122. Springer, 2012. doi:10.1007/978-3-642-33122-0_9.

19 Daniel Lemire. Ewahboolarray, 2026. Accessed: 2026-05-09. URL: https://github.com/lemire/EWAHBoolArray.

20 Alitzel López Sánchez and Manuel Lafond. Galled perfect transfer networks. In *RECOMB International Workshop on Comparative Genomics*, pages 24–43. Springer, 2024. doi:10.1007/978-3-031-58072-7_2.

21 Diego Mallo, Leonardo de Oliveira Martins, and David Posada. Simphy: phylogenomic simulation of gene, locus, and species trees. *Systematic biology*, 65(2):334–344, 2016.

22 Megan Owen. Bhvtreespace, 2026. Accessed: 2026-05-09. URL: https://github.com/megan-owen/BHVtreespace.

23 Megan Owen and J Scott Provan. A fast algorithm for computing geodesic distances in tree space. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 8(1):2–13, 2010. doi:10.1109/TCBB.2010.3.

24 Andy Purvis. A composite estimate of primate phylogeny. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, 348(1326):405–421, 1995.

25 Charles Semple, Mike Steel, et al. *Phylogenetics*, volume 24. Oxford University Press on Demand, 2003.

**26** Leo Van Iersel, Steven Kelk, Regula Rupp, and Daniel Huson. Phylogenetic networks do not need to be complex: using fewer reticulations to represent conflicting clusters. *Bioinformatics*, 26(12):i124–i131, 2010.

**27** Mathias Weller. Listing conflicting triples in optimal time. In *16th Cologne-Twente Workshop on Graphs and Combinatorial Optimization*, page 144, 2018.

**28** Christian M Zmasek and Sean R Eddy. A simple algorithm to infer gene duplication and speciation events on a gene tree. *Bioinformatics*, 17(9):821–828, 2001. doi:10.1093/BIOINFORMATICS/17.9.821.


## A Proofs for Section 3 (Computing incompatibility graphs)

▶ **Theorem 6.** *Algorithm 1 correctly computes the incompatibility graph $G_{T,T'}$ of two trees $T, T'$ with the same leaves in time $O(n + d)$, where $n$ is the number of leaves of the input trees, and $d$ is the number of edges of $G_{T,T'}$.*

**Proof.** Let $T, T'$ be the two input trees. We first consider the time complexity of the algorithm, then prove correctness afterwards.

**Running time analysis.** Let us first observe that for any two nodes $u, v \in V(T)$, we have $u \preceq_T v$ if and only if $lca_T(u, v) = v$. Thus constant time lca queries allow checking whether $u \preceq_T v$ in time $O(1)$. We can also check if $u \prec_T v$ by simply ensuring that $u \preceq_T v$ and $u \neq v$. The same holds for ancestry relations in $T'$.

Let us now consider the pre-processing steps of the algorithm. Pre-processing for $O(1)$ lca queries, constructing $\vec{\mu}$ and $\overleftarrow{\nu}$, and pre-processing for $O(1)$ $diff_anc$ queries are already known to be doable in time $O(n)$. For the graph $G$, we assume that it is stored as an adjacency list. Specifically, we assume that each node in $V(T) \cup V(T')$ is assigned a unique integer between 1 and $|V(T) \cup V(T')|$, which is $O(n)$. To initialize $G$, we create an array of length $|V(T)|$, with the $i$-th element intended to contain the list of neighbors of the $i$-th vertex of $T$, with each element of the array consisting of an empty list initially. This initialization takes time $O(n)$, and we assume that inserting an element in the $i$-th list takes constant time. However, we do *not* assume that checking whether an element is present in the $i$-th list takes constant time, which is why we use the global array $Q$. Speaking of which, this array also take time $O(n)$ to initialize.

Let us next argue that any edge $\{v, x\}$ gets added at most once during the course of the whole algorithm. This is because edges incident to $v$ are only added when we iterate on $v$ in the main procedure. During that iteration, we always check that $Q[x] \neq v$ before adding $\{v, x\}$ to $G$, and if we do add the edge we set $Q[x] = v$ right after, which ensures that the edge cannot be added a second time later on. Thus we add every edge of the incompatibility graph once, and so the number of times that we go through the instruction "Add $\{v, x\}$ to $E(G)$" in either subfunction is exactly $|E(G)| = d$. In particular, this implies that in total, throughout the execution of the whole algorithm, we spend a total time of $O(d)$ inside the loop of line 16 in *check_between*, because each iteration adds an edge and each edge is added once (and the conditions and instructions of the loop are easily seen to be doable in constant time).

With this in mind, we now focus on the time needed to call $check_between(v, G)$ for a node $v \in V(T)$. We make $|ch_T(v)|$ iterations of the main loop that iterates on the children of $v$. For one such child $w$, consider the loop "*while* $x \neq \perp$ and $\overleftarrow{\nu}(x) \preceq_T v$" on line 14. We claim that for any $v$ and $w$, we spend time $O(1)$ in this loop. Before the first iteration, $x = \vec{\mu}(w)$, and so $\overleftarrow{\nu}(x)$ must be $w$ or one of its ancestors (this is because $\vec{\mu}(w) = x$ implies that $C_{T'}(x)$ contains all of $C_T(w)$, and $\overleftarrow{\nu}(x)$ is thus an ancestor of all elements of $C_T(w)$).

This means that before entering the loop, either $\overleftarrow{\nu}(x)$ is $w$, or $v$, or a strict ancestor of $v$. If $\overleftarrow{\nu}(x) = w$, then in the next iteration, the value of $\overleftarrow{\nu}(x)$ must change by the definition of $diff_anc$, and therefore after 0 or 1 iteration we know that $\overleftarrow{\nu}(x)$ is $v$ or one of its strict ancestors (or $\perp$ and we exit the loop). Another iteration is required at this stage only if $\overleftarrow{\nu}(x) = v$. The next iteration after that ensures that $v \prec_T \overleftarrow{\nu}(x)$ or $\overleftarrow{\nu}(x) = \perp$, so we make at most two iterations of this loop, each taking $O(1)$ time by our pre-processing for fast $diff_anc$ queries. Thus if we omit the time spent inside the loop on line 16, *check_between* takes constant time per child $w$ of $v$ and therefore a total time of $O(|ch_T(v)|)$ on input $v$ (this includes the time we spend to check the conditions of the loop on line 16 without entering it). Overall, adding in the time $O(d)$ spent in the loop of line 16, we total time spent in *check_between* over all $v \in V(T)$ is

$$O(d + \sum_{v \in V(T)} |ch_T(v)|) = O(d + n),$$

which is because $\sum_{v \in V(T)} |ch_T(v)|$ is just the number of edges of $T$, and this number is $O(n)$.

Now let us consider the time spent in *check_incomp_child*. For $w \in V(T)$, we denote by $deg_G(w)$ the number of edges incident to $w$ in $G$. Again, we make $|ch_T(v)|$ iterations of its main loop. For each child $w$ of $v$, we first go through the loop on line 23, which iterates through the $deg_G(w)$ edges $\{w, x\}$ incident to $w$ in $G$ (note, the enumeration takes time $O(deg_G(w))$ since $G$ is implemented as an adjacency list). For each such edge $\{w, x\}$, the check for $v \prec_T \overleftarrow{\nu}(x)$ and $Q_a[x] \neq v$, and if needed adding $\{v, x\}$ to $E(G)$ and putting $Q_a[x]$ all take $O(1)$ time. Therefore, the loop on line 23 takes time $O(deg_G(w))$ per child $w$. For input $v$, this results in a time of $O(|ch_T(v)| + \sum_{w \in ch_T(v)} deg_G(w))$, which overall across $T$ is

$$O\left(\sum_{v \in V(T)} \left(|ch_T(v)| + \sum_{w \in ch_T(v)} deg_G(w)\right)\right) = O(n + d).$$

To see why, notice that $\sum_{v \in V(T)} |ch_T(v)|$ is the number of edges of $T$ and thus $O(n)$, and in the sum $\sum_{v \in V(T)} \sum_{w \in ch_T(v)} deg_G(w)$, we have that for any $w \in V(T)$, the term $deg_G(w)$ is added at most once, which is when the first summation over $v$ is iterating on the parent $v$ of $w$.

By summing the $O(n)$ time spent in pre-processing, plus the $O(n + d)$ time spent in *check_between*, plus the $O(n + d)$ time spent in *check_incompat_child*, we get a running time of $O(n + d)$.

**Correctness.** Let us now argue correctness. We prove by induction on the postordering of $V(T)$ that for each $v \in V(T)$, the algorithm adds an edge $\{v, x\}$ if and only if $v, x$ are incompatible. Note that this statement allows $v$ to be a leaf. As a base case, suppose that $v$ is a leaf of $T$. Then $v$ is not incompatible with any node of $T'$. The functions *check_between* and *check_incomp_child* are not even called on input $v$, so the algorithm adds no edge incident to $v$ in $G$, which is correct.

Now assume that $v$ is not a leaf and assume by induction that in $G$, the edges incident to each child $w$ of $v$ are correct.

Suppose that the algorithm adds an edge $\{v, x\}$. Assume first that the edge is added in the call to *check_between* on input $v$, while iterating over some child $w$ of $v$. That iteration starts with $x = \vec{\mu}(w)$. At this moment, Observation 3 implies that $C_{T'}(x) \cap C_T(w) \neq \emptyset$, which in turn implies $C_{T'}(x) \cap C_T(v) \neq \emptyset$. Then the loop on line 14 maintains that property, since $x$ is updated to become one of its ancestors (unless we get $x = \perp$, but we exclude this

case since it would become impossible to add an edge in the other loop of line 16). The same property also holds across every iteration of loop on line 16, so every value of $x \neq \perp$ encountered in the iteration on $w$ satisfies $C_{T'}(x) \cap C_T(v) \neq \emptyset$.

When we exit the loop on line 14, since we assume $x \neq \perp$ we get that $\overleftarrow{\nu}(x) \preceq_T v$ is false after the loop. Since $C_{T'}(x) \cap C_T(v)$ is non-empty, $\overleftarrow{\nu}(x)$ is an ancestor of some descendant of $v$ and thus $\overleftarrow{\nu}(x), v$ cannot be incomparable. It follows that $v \prec_T \overleftarrow{\nu}(x)$. The latter property is maintained after every iteration of the loop on line 16, when $x$ is updated to its parent, because $\overleftarrow{\nu}(x) \preceq_T \overleftarrow{\nu}(parent_{T'}(x))$ (as long as $x \neq \perp$). Then, if we add an edge on line 17, the condition $x \prec_{T'} \vec{\mu}(v)$ is verified explicitly. Thus if we reach that line, we have $C_{T'}(x) \cap C_T(v) \neq \emptyset$, $v \prec_T \overleftarrow{\nu}(x)$, and $x \prec_{T'} \vec{\mu}(v)$. These are the conditions of Lemma 4, and $v, x$ are incompatible, as desired.

So assume that the algorithm adds $\{v, x\}$ in a call to *check_incomp_child*, while iterating over some child $w$ of $v$. Thus $v$ has a child $w$ such that $\{w, x\} \in E(G)$, and so by induction $w, x$ are incompatible, and $v \prec_T \overleftarrow{\nu}(x)$ if checked explicitly. By Lemma 5 $v, x$ are incompatible and adding $\{v, x\}$ is correct. Therefore, all edges added by the algorithm are correct.

Conversely, we argue that if $v, x'$ are incompatible, then the algorithm adds the edge $\{v, x'\}$ (we use $x'$ here to distinguish it from the variable $x$ used by the algorithm). Note for later reference that for any $x \in V(T')$, $\{v, x\}$ is added by the algorithm if and only $Q[x]$ is set to $v$ at some point (because when entering *check_between* on input $v$, no previous iteration could have put the value of $v$ in $Q$, and then $Q[x]$ is set to $v$ precisely after adding $\{v, x\}$ in either subroutine).

Now, by Lemma 5, the incompatibility of $v, x'$ implies that $v \prec_T \overleftarrow{\nu}(x')$ and $v, x'$ satisfy one of the two conditions of the lemma. Suppose first that condition (a) is true, i.e., $v$ has a child $w$ such that $\vec{\mu}(w) \preceq_{T'} x' \prec_{T'} \vec{\mu}(v)$. We claim that $\{v, x'\}$ gets added by *check_between*. The main difficulty of the proof is to argue that the $Q[x] \neq v$ condition does not make us miss edges.

Let $w_1, w_2, \ldots, w_k$ be the children of $v$, in the order that the function *check_between* iterates on them. Note that for $i = 1, \ldots, k$, when iterating on $w_i$, all the values of $x$ encountered in *check_between* are ancestors of $\vec{\mu}(w_i)$. Thus after iterating on $w_i$, having $Q[x] = v$ is only possible if $x$ is an ancestor of at least one of $w_1, \ldots, w_i$. Now let $j$ be the minimum index such that $x'$ is an ancestor of $\vec{\mu}(w_j)$ (such a $j$ exists since $\vec{\mu}(w) \preceq_{T'} x'$ for some $w$). Recall that by Lemma 5, we know that $v \prec_T \overleftarrow{\nu}(x')$. When iterating on $w_j$, the loop of $diff_anc$ reassignments on line 14 starts with $x = \vec{\mu}(w_j)$ and will stop at some node $x$ that descends from $x'$ (with equality possible). Since $x \preceq_{T'} x'$, none of $w_1, \ldots, w_{j-1}$ has $x$ as an ancestor (otherwise it would have $x'$ as an ancestor), and therefore $Q[x] \neq v$. The same is true for any ancestor of $x$. Moreover, we have $x' \prec_{T'} \vec{\mu}(v)$ by Lemma 4, and thus $x \prec_{T'} \vec{\mu}(v)$ as well, and the same holds for every node on the path between $x$ and $x'$. It follows that when iterating on $w_j$, the algorithm will enter the loop of line 16 for $x$ and all of its ancestors up to $x'$, and therefore $\{v, x'\}$ will correctly get added to $E(G)$.

Suppose instead that condition (b) is satisfied, i.e., $v$ has a child $w$ such that $w, x'$ are incompatible. If $\{v, x'\}$ gets added by *check_between*, we are done, so assume otherwise. Consider the call to *check_incomp_child* on input $v$, when we iterate on $w$. The edge $\{w, x\}$ is present by induction and the algorithm will enter the loop of line 23 on edge $\{w, x\}$. If $Q[x] = v$, the edge was added at an earlier moment and we are done. Since $v, x'$ are incompatible, $v \prec_T \overleftarrow{\nu}(x')$ by Lemma 5. We therefore enter the "if" statement and add $\{v, x'\}$, as desired.

We have thus shown that $v, x$ are incompatible if and only if the algorithm adds $\{v, x\}$ to $G$, which concludes the correctness proof. ◀

▶ **Proposition 8.** *Let $U, U'$ be two unrooted trees with $L(U) = L(U') = \{1, 2, \ldots, n\}$ with $n \geq 3$, and let $T, T'$ be the rooted trees obtained from $U$ and $U'$, respectively, by assigning the neighbor of leaf 1 as the root in both trees.*

*Let $uv \in E(U)$ and $xy \in E(U')$, where $u$ is the parent of $v$ in $T$ and $x$ is the parent of $y$ in $T'$. Then $S_U(uv)$ and $S_{U'}(xy)$ are compatible splits if and only if $C_T(v)$ and $C_{T'}(y)$ are compatible clusters.*

**Proof.** First note that the condition $n \geq 3$ ensures that the neighbor of leaf 1 in both $U$ and $U'$ is not a leaf, and thus that the rootings are valid (in particular, since no node of $U$ and $U'$ has degree 2, the internal nodes of $T$ and $T'$ all have at least two children).

Let $A|B$ be the split of $uv$ in $U$, with $B = C_T(v)$, and let $C|D$ be the split of $xy$ in $U'$, with $D = C_{T'}(y)$. If $B$ contains 1, then $B = \{1\}$ because 1 is a child of the root. In that case, one of $B \cap C$ or $B \cap D$ is empty because one of $C$ or $D$ does not contain 1, and the two splits are compatible. Moreover, $B$ and $D$ are compatible clusters because a cluster with a single element is compatible with any cluster. Thus our statement holds when $B$ contains 1. Using the same arguments, it also holds if $D$ contains 1.

Assume from now on that $B$ and $D$ both do not contain 1. Suppose that $A|B$ and $C|D$ are compatible splits. Then by Proposition 7, one of the intersections $A \cap C, A \cap D, B \cap C, B \cap D$ is empty. Note that $A \cap C \neq \emptyset$ since both $A$ and $C$ contain the leaf 1, because $A = L(U) \setminus B$ and $C = L(U') \setminus D$. If $A \cap D = \emptyset$, then every element of $D$ must be in $B$, and $B, D$ are compatible clusters by Proposition 1 since $D \subseteq B$. If $C \cap B = \emptyset$, then likewise $B \subseteq D$ and $B, D$ are compatible clusters. Finally, if $B \cap D = \emptyset$, then $B, D$ are compatible clusters directly by Proposition 1. In all cases, $A|B, C|D$ being compatible implies that $B = C_T(v)$ and $D = C_{T'}(y)$ are compatible.

Conversely, suppose that $B, D$ are compatible clusters. By Proposition 1, either $B \cap D = \emptyset$, or $B \subseteq D$, or $D \subseteq B$. If $B \cap D = \emptyset$, then $A|B, C|D$ are compatible splits by Proposition 7. If $B \subseteq D$, then $B \cap C = \emptyset$ and the two splits are compatible, and if $D \subseteq B$ then $A \cap D = \emptyset$ and the two splits are again compatible. This concludes the proof. ◀

▶ **Proposition 9.** *Let $T, T'$ be two trees and let $\tilde{T}, \tilde{T}'$ be defined as above. Then $|V(G_{\tilde{T},\tilde{T}'})| \leq |V(G_{T,T'})| + 2$. Moreover, $ux$ is an edge of $G_{T,T'}$ if and only if $ux$ is an edge of $G_{\tilde{T},\tilde{T}'}$.*

**Proof.** The bound on the number of vertices is obvious because $\tilde{T}$ has at most one more internal node than $T$, and $\tilde{T}'$ has at most one more internal node than $T'$. For the other statement, suppose first that $u \in I(T)$ and $x \in I(T')$. Notice that $C_T(u) = C_{\tilde{T}}(u)$ and $C_{T'}(x) = C_{\tilde{T}'}(x)$. Therefore for such pairs of vertices, $ux \in E(G_{T,T'})$ if and only if $E(G_{\tilde{T},\tilde{T}'})$.

Now suppose that $u \in I(\tilde{T}) \setminus I(T)$. Then it must be that $u = \rho(\tilde{T})$. By construction, we have $C_{\tilde{T}}(u) = L(T) \cup L(T')$. Also, for any $x \in I(\tilde{T}')$, we have $C_{\tilde{T}'}(x) \subseteq L(T) \cup L(T') = C_{\tilde{T}}(u)$. Thus $u$ and $x$ are compatible. Therefore, $u$ is an isolated vertex in $G_{\tilde{T},\tilde{T}'}$. Using a symmetric argument, we get that any $x \in I(\tilde{T}') \setminus I(T')$ must be the root of $\tilde{T}'$ and an isolated vertex in $G_{\tilde{T},\tilde{T}'}$. It follows that the edge sets of $G_{T,T'}$ and $G_{\tilde{T},\tilde{T}'}$ are identical. ◀

# FBApro: A Fast, Simple Linear Transformation for Diverse Metabolic Modeling Tasks

**Ariel Bruner**
Princeton University, NJ, USA

**Mona Singh**
Princeton University, NJ, USA

## Abstract

Constraint-based metabolic modeling is the predominant framework for simulating cellular metabolism. The central assumption of these models is that metabolism operates at a steady state, meaning that the production and consumption rates of each metabolite are balanced. This assumption imposes linear constraints on the fluxes of biochemical reactions. Flux Balance Analysis (FBA), a fundamental method in the field, is formulated as an optimization problem maximizing a cellular objective (e.g., growth) over the resulting linear subspace of steady state fluxes. Many other methods in the field are expressed either as a modification to FBA, or use FBA as a black box within an algorithm. Here, we propose a general alternative to optimization called FBApro. For any given vector of reference fluxes, FBApro finds the closest flux vector within the steady-state subspace, and accounts for both partially given reference fluxes and exact constraints on reactions. While FBApro is the solution to a quadratic program, we show that it can be implemented as a single linear operation using orthogonal projections to corresponding affine spaces and sets of linear equations. The overall approach is computationally efficient, does not require a cellular objective, and is easy to implement. We formally derive the closed-form expressions for FBApro and simpler variants, and validate it on both synthetic and real cancer cell line data.

**Code availability.** The code implementing FBApro is available at `https://github.com/Singh-Lab/FBApro`. All code required to reproduce the figures in the paper is available, although the data used must be sourced separately. The repository also contains toy models and examples.

**2012 ACM Subject Classification** Applied computing → Systems biology

**Keywords and phrases** metabolic modeling, flux balance analysis, constraint-based metabolic modeling

**Digital Object Identifier** 10.4230/LIPIcs.WABI.2026.26

**Related Version** *Preprint*: `https://arxiv.org/abs/2601.14577`

**Supplementary Material** *Software (Source Code)*: `https://github.com/Singh-Lab/FBApro` [5] archived at `swh:1:dir:56f800231d0dae5268df2adbab6fd5599d0a376f`

**Funding** This work was supported in part by the Ludwig Institute for Cancer Research, Princeton Branch and NIH grant R01-CA208148.

**Declaration on GenAI/LLM use** LLM tools were used for editing, proof reading, and generation of graphics.

## 1 Introduction

Cellular metabolism is a complex, tightly regulated process that underlies the behavior of biological systems. Modeling metabolism requires characterizing both the biochemical capabilities of organisms, as well as developing computational methods to simulate these processes under specific conditions. Constraint-Based Metabolic Modeling (CBMM) has emerged as a widely-used framework for such simulations. It assumes a steady-state regime in which


© Ariel Bruner and Mona Singh;
licensed under Creative Commons License CC-BY 4.0

metabolite concentrations remain constant and reaction fluxes must balance. Mathematically, the steady-state regime is expressed as the kernel of a stoichiometric matrix representing the metabolic model (see Figure 1, and Appendix Section A.1 for more details).

CBMM has a long history, with theoretical foundations laid more than 40 years ago. Its most fundamental method is Flux Balance Analysis (FBA) [24, 23, 11], which formulates a linear program to capture steady-state flux constraints, individual reaction bounds and an organism-level objective. Large-scale sequencing efforts have enabled the construction of genome-scale metabolic models for numerous organisms [22], and FBA has been instrumental in predicting the metabolic phenotypes of unicellular organisms under various growth conditions [9, 26, 18, 19]. Extensions to FBA have also enabled FBA-based predictions of gene knockout effects (e.g., MoMA [27] and ROOM [28]).

Modeling of multi-cellular organisms remains challenging due to heterogeneity across cell types, unclear metabolic objectives, and limited knowledge of *in vivo* metabolite availability. These problems are even more pronounced in applications of CBMMs to cancer metabolism, where altered metabolism plays a significant role [6]. The unique challenges of multicellular metabolic modeling, together with the increased availability of 'omics data, have led to the development of numerous methods integrating 'omics data with metabolic models. Such data can help tailor CBMMs to specific cell types, tissues, and disease contexts by identifying which metabolic genes and pathways are likely active, thereby helping to constrain the space of feasible fluxes and improving context-specific predictions. Gene expression (GE) data are the most widely available 'omics data, and many algorithms integrate GE with metabolic models to construct context-specific networks or infer context-specific reaction fluxes (e.g., GIMME [4], iMAT [29], MBA [15], INIT [1]), including extensions to incorporate single-cell RNA-seq data (e.g., Compass [30]). Most of these methods formulate optimization problems that incorporate both steady-state constraints and data fidelity terms, often resulting in large linear or mixed-integer programs, sometimes approximated by greedy heuristics. Scaling these approaches to large cohorts or single-cell datasets poses substantial computational challenges.

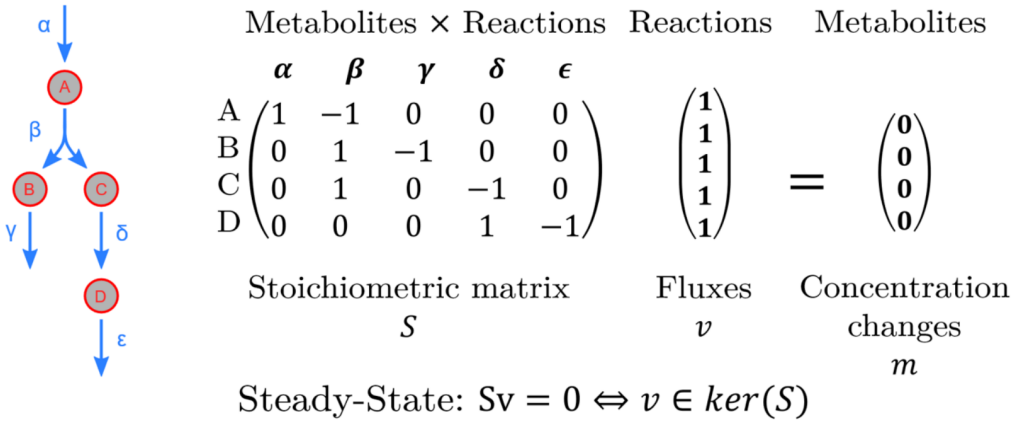



**Figure 1 Left:** A metabolic model. Each node corresponds to a metabolite, and each hyperedge corresponds to a reaction. For example, reaction $\beta$ consumes metabolite $A$ and produces metabolites $B$ and $C$. **Right:** The steady-state assumption in CBMM. Each column in the stoichiometric matrix $S$ represents the metabolite changes induced by one reaction (e.g., see the column corresponding to the reaction $\beta$). Multiplying $S$ by the flux vector $v$, whose entries represent reaction rates, gives the net rate of change in metabolite concentrations. The steady state assumption in CBMM is that metabolite concentrations do not change over time; therefore feasible flux vectors satisfy $Sv = 0$.

In parallel, advances in deep learning have enabled breakthrough advances in other areas of computational biology (e.g., [16]), and highlight the value of domain-specific methods that admit closed-form differentiable implementations that can be embedded directly into gradient-based machine learning models. However, to our knowledge, all available CBMM algorithms are based on solving optimization problems, particularly linear and mixed-integer linear programs, which are not easily integrated into end-to-end differentiable pipelines.

Here we present FBApro (short for FBA projection), a closed-form linear operation for solving the following general reference-flux agreement problem: given a reference flux vector $v$ for a metabolic model with $r$ reactions and steady-state space $ker(S) \subseteq \mathbb{R}^r$, a subset of high-confidence reactions $H \subseteq [r]$ and a disjoint subset of medium-confidence reactions $M \subseteq [r]$, find a steady-state flux vector that agrees with $v$ on $H$ and minimizes the $L_2$ distance to $v$ on $M$. This formulation models settings in which the reference vector $v$ combines heterogeneous sources of information: for example, we may have experimentally-derived flux measurements for reactions in $H$ (typically known for only a small number of reactions) and lower-confidence activity estimates for gene-associated reactions, inferred from gene expression or other 'omics data (not all reactions in metabolic models are associated with known genes). FBApro is presented as a projection onto an affine space determined by $S, M, H$ and the values of $v$ on $H$. Although this affine space depends on the reference flux input vector $v$, we show that the resulting projection can nevertheless be expressed as a linear operator on $\mathbb{R}^r$ that is independent of $v$. Thus, for fixed $S$, $M$, and $H$, the projection matrix can be precomputed once for a given experimental setup and then applied by matrix multiplication to any number of reference flux vectors. FBApro is therefore orders of magnitude faster than repeated applications of linear or quadratic programming based approaches when amortized over many inputs, while additionally being fully differentiable as a function of $v$. This amortized setting arises naturally in large-scale studies of human cancer cell lines or samples, where GE data are available across many biological contexts, direct flux measurements are sparse or unavailable, and the goal is to infer context-specific flux states across many samples.

FBApro is related to previous quadratic programming formulations, such as MoMA [27], which minimizes the distance of a knockout model steady-state flux from a wild-type reference flux distribution, and the approach of [12] for fitting partially measured flux data. In contrast to these methods, FBApro is expressible as a simple orthogonal projection, at the cost of foregoing individual reaction bounds. To our knowledge, a projection-based method for solving the reference-flux agreement problem has not previously been formally described, proposed as an alternative to FBA-derived optimization methods, or validated on empirical data. Orthogonal projections have been used in the flux sampling module of the cobrapy metabolic modeling library [8] for the special case of $M = [r]$, where they serve as a computational step for mapping sampled points to the steady-state space. In contrast, we show that projection can be used as the basis for solving more general reference-agreement problems involving both equality-constrained and medium-confidence reaction sets. We present simplified representations for FBApro for this case and the other special cases $H = \varnothing$ and $M \cup H = [r]$, with some cases admitting multiple equivalent expressions.

We illustrate the use of FBApro on an example "toy" model, and validate it on both simulated and cancer cell line data. On simulated data, we first compare the running time of FBApro to FBA, iMAT [29], and MoMA [27], and find that FBApro is orders of magnitude faster. We compare FBApro to FBA as it is the *de facto* method for metabolic modeling and has been used in various ways to incorporate GE values [7, 17, 20]; to iMAT as it is a widely used method for integrating GE with metabolic models to uncover steady state fluxes that are "consistent" with the GE data; and to MoMA as it solves a similarly phrased problem to FBApro using the traditional optimization approach. The aforementioned methods also

have an available implementation written in Python. We then consider simulations where exact, noisy and missing data are derived from steady state fluxes, and demonstrate that FBApro is competitive with previous approaches in recapitulating the known steady state fluxes, while requiring a fraction of the running time. Finally, we consider cancer cell-line GE and reaction flux data to show that FBApro surpasses or has competitive performance in imputing unmeasured fluxes and mapping GE data to the steady-state space while predicting corresponding unseen data. Overall, our results suggest that FBApro is a flexible, modular and effective approach for high-throughput and large-scale metabolic modeling.

**Organization of remainder of the paper.** In Section 2, we give formal definitions of the variants of FBApro and state the theorems that allow us to derive the algorithms underlying FBApro. In Section 3, we prove the theorems and derive the expressions for variants. In Section 4, we show how FBApro can be utilized to integrate data into a model, using a "toy" metabolic model. In Section 5, we demonstrate the performance of FBApro on simulated and real empirical data, and give some concluding remarks in Section 6.

# 2 Problem Statements

## 2.1 Definitions and Notations

Let $m, r \in \mathbb{N}$ be the number of metabolites and reactions in a metabolic model, and $S \in \mathbb{R}^{m \times r}$ be the stoichiometric matrix of the model, such that a unit of flux in reaction $j$ produces $S_{i,j}$ units of metabolite $i$ (or consumes, if negative). The steady-state space of the model is the set of flux assignments for which each metabolite has zero net production; i.e., $ker(S) = \{v \in \mathbb{R}^r | Sv = 0\}$. As an alternative representation for the steady-state space, let $l = dim(ker(S))$ and let $A \in \mathbb{R}^{r \times l}$ be a matrix whose columns form an orthonormal basis of $ker(S)$, so that $col(A) = ker(S)$ and $A^T A = I$, computable from $S$ using a Singular Value Decomposition (SVD).

Our methods use the Moore-Penrose inverse for projections and solutions for systems of linear equations. For a matrix $X$, let $X^+$ denote its Moore-Penrose inverse, or pseudoinverse (also computable from $X$ using SVD). We also make use of vector restrictions, and associated restriction matrices: for a vector $v$ and index set $Q = \{q_1, \ldots, q_{|Q|}\}$, denote by $v_Q = (v_{q_1}, \ldots, v_{q_{|Q|}})^T$ the restriction of $v$ to $Q$, and let $P_Q \in \{0,1\}^{|Q| \times r}$ be the restriction matrix to $Q$,

$$(P_Q)_{i,j} = \begin{cases} 1 & j = q_i \\ 0 & \text{otherwise,} \end{cases}$$

such that $\forall v. P_Q v = v_Q$. Similarly, for a matrix $X \in \mathbb{R}^{a \times b}$ and index sets $\mathcal{I} \subseteq [a], \mathcal{J} \subseteq [b]$, we denote the submatrix restricted to these sets by $X_{\mathcal{I},\mathcal{J}} = \{X_{i,j} | i \in \mathcal{I}, j \in \mathcal{J}\}$. Denote by $M, H \subseteq [r]$ the indices of medium and high confidence reactions, respectively, with an arbitrary order over the sets. Let $B \in \mathbb{R}^{r \times (r - |H|)}$ be the matrix whose columns are the standard basis vectors $\{e_i | i \notin H\}$ of $\mathbb{R}^r$, spanning all flux vectors with the reactions of $H$ carrying no flux.

## 2.2 FBApro: General case

The main problem we solve is the general problem of finding a steady-state flux $x \in \mathbb{R}^r$ given $S$ approximating a given reference flux $v \in \mathbb{R}^r$ for reactions with indices in $M \subseteq [r]$ and equaling $v$ for reactions with indices in $H \subseteq [r]$, with $H \cap M = \varnothing$. Our problem can be formulated as:

▶ **Problem 1.**

$$\begin{aligned} FBApro(S, M, H, v) = argmin_{x \in \mathbb{R}^r} \quad & \|x_M - v_M\|_2^2 \\ subject\ to \quad & Sx = 0 \\ & x_H = v_H, \end{aligned}$$

where *argmin* refers to any solution minimizing the distance. An algorithm that solves FBApro in this most general form derives from the following theorem:

▶ **Theorem 1.** *With $P := P_M$ the restriction matrix to $M$, and $B$ spanning all fluxes in reactions not in $H$, define*

$$D = AA^T + BB^T, \qquad C = [BB^T D^+ A : AA^T D^+ B], \qquad E = C(PC)^+ P,$$

*with $:$ denoting concatenation of columns. Then, if a solution to Problem 1 exists,*

$$FBApro(S, M, H, v) = \left(E(I_r - AA^T D^+) + AA^T D^+\right) v.$$

The proof of this Theorem is given in Subsection 3.3. In practice, this theorem shows that, given $v$, FBApro($v$) can be simply expressed as a multiplication of $v$ by a precomputed matrix[1]. Computing this matrix requires a constant number of matrix multiplication, addition and concatenation operations, as well as a constant amount of SVD computations, all for matrices of $O(max\{r, m\})$ rows and columns. Typical metabolic models have numbers of reactions and metabolites on the same order, and so assuming $m \in \Theta(r)$ the entire preprocessing has time complexity $O(r^3)$, and theoretically lower if fast matrix multiplication is considered. Each subsequent application of FBApro for different reference value vectors has a time complexity of $O(r^2)$.

Linear programs like FBA are typically solved with the simplex method, which has a worst-case exponential complexity but is polynomial in practice, and provably polynomial methods exist admitting a complexity of $O(r^3)$, similarly improved by fast matrix multiplication. Convex quadratic programs like MoMA also have a cubic complexity, in contrast to mixed-integer programs like iMAT which are NP-hard. Note that unlike FBApro, there is no improvement in amortized time complexity gained by solving these programs for multiple samples.

## 2.3 FBAproBasic: Noisy Flux Estimates for All Reactions

Before proving our main result, we also consider special cases of FBApro, the simplest of which is formalized next.

▶ **Problem 2.**

$$\begin{aligned} FBAproBasic(S, v) = argmin_{x \in \mathbb{R}^r} \quad & \|x - v\|_2^2 \\ subject\ to \quad & Sx = 0, \end{aligned}$$

corresponding to $M = [r]$. This formulation is similar in spirit to the optimization problem posed by MoMA [27], differing in the exclusion of reaction bounds $l \leq v \leq u$, and the typical MoMA use case of simulating knockout behavior with the constraint $v_i = 0$ for some reaction $i$. Since a knockout can also be represented either by setting $l_i = u_i = 0$ or

[1] We will sometimes refer to FBApro as FBAproFull to differentiate it from other simpler versions of the algorithm.

by considering a stoichiometric matrix with column $i$ removed, only the reaction bounds meaningfully differentiate the two problems. However, unlike MoMA, FBAproBasic admits a closed-form solution using an orthogonal projection, as described in the next theorem:

▶ **Theorem 2.**

$$FBAproBasic(S, v) = AA^T v = (I - S^+ S)v.$$

FBAproBasic is equivalent to an orthogonal projection to a linear subspace, a standard linear algebra operation. Nevertheless, for completeness and as a "warm up" for the other proofs, we include in Appendix Section B a standalone proof for Theorem 2, relying only on basic linear algebra and the properties of the pseudoinverse.

## 2.4 FBAproPartial: Handling Unknown Flux Values

The following case corresponding to $H = \varnothing$ is not directly represented as an orthogonal projection, and to our knowledge has not been studied before:

▶ **Problem 3.**

$$\begin{aligned} FBAproPartial(S, M, v) = argmin_{x \in \mathbb{R}^r} \quad & \|x_M - v_M\|_2^2 \\ \textit{subject to} \quad & Sx = 0, \end{aligned}$$

for which we show:

▶ **Theorem 3.**

$$FBAproPartial(S, M, v) = A(P_M A)^+ P_M v.$$

Thus, as with general FBApro and the simple FBAproBasic, the algorithm for finding the flux vector satisfying our constraints corresponds to precomputation of a matrix based on the metabolic model and knowledge of which reactions mid-confidence, followed by a simple matrix multiplication involving the input flux vector.

## 2.5 FBAproFixed: Handling Exact Flux Values

The case corresponding to $M \cup H = [r]$ leads to the following problem:

▶ **Problem 4.**

$$\begin{aligned} FBAproFixed(S, H, v) = argmin_{x \in \mathbb{R}^r} \quad & \|x - v\|_2^2 \\ \textit{subject to} \quad & Sx = 0 \\ & x_H = v_H, \end{aligned}$$

This problem can be thought of as an orthogonal projection to the affine space $(v + col(B)) \cap ker(S)$, if a solution exists and the intersection is not empty. We first derive an expression for FBAproFixed using basic linear algebra:

▶ **Theorem 4.** *Define $H^c = [r] \setminus H$ and define $Q \in \mathbb{R}^{r \times r}$ such that its restriction submatrices are*

$$\begin{aligned} Q_{H,H} &= I_{|H|} \\ Q_{H,H^c} &= 0 \\ Q_{H^c,H} &= -\left(S_{[m],H^c}\right)^+ S_{[m],H} \\ Q_{H^c,H^c} &= \left(I_{|H^c|} - \left(S_{[m],H^c}\right)^+ S_{[m],H^c}\right). \end{aligned}$$

*If a solution to Problem 4 exists, then $FBAproFixed(S, H, v) = Qv$.*

We prove this derivation in the Appendix Section C. Before proving our main theorem, we draw on [25] to derive the following alternative representation:

▶ **Theorem 5.** *As previously defined, let*

$$D = AA^T + BB^T, \qquad C = [BB^T D^+ A : AA^T D^+ B],$$

*If a solution to Problem 4 exists, then*

$$FBAproFixed(S, H, v) = \left(CC^+ \left(I_r - AA^T D^+\right) + AA^T D^+\right) v.$$

The proof is given in Subsection 3.2, and is followed by the proof of the main theorem in Subsection 3.3.

## 3 Deriving FBApro

Before proving Theorem 1 in Section 3.3, which gives the formula for solving FBApro in its most general form, we first derive formulas for two simpler cases. In Section 3.1, we consider FBAproPartial, where the reference flux contains medium-confidence, approximate reaction values that we aim to match as closely as possible, as well as reactions for which no reference information is available. In Section 3.2, we consider FBAproFixed, where the reference flux contains high-confidence reaction values that must be maintained exactly, with the remaining values optimized over. The proof techniques developed for these two subproblems are then combined to derive the general FBApro solution.

### 3.1 FBAproPartial

Recall that $FBAproBasic(S, v) = AA^T v = (I - S^+ S)v$. While FBAproBasic can alternatively be computed using either $S$ or $A$, representing the steady-state space as $col(A)$ allows for a straightforward representation of the steady-state restrictions using the restriction matrix $P := P_M$, yielding our single representation for FBAproPartial:

**Proof of Theorem 3.** Recall that $\forall x.Px = x_M$. Then $\{x_M | x \in col(A)\} = col(PA)$ and so

$$\min_{x \in col(A)} \|v_M - x_M\|_2^2 = \min_{y \in col(PA)} \|Pv - y\|_2^2.$$

Since $col(PA)$ is a linear space spanned by $PA$, we know from the proof of Theorem 2 that a distance minimizing $y$ in the space can be expressed as $y = (PA)(PA)^+(Pv)$. Finally, we want an $x \in \text{col}(A)$ satisfying $Px = y$, but we already have the expression $y = P(A(PA)^+ Pv)$, yielding $x = A(PA)^+ Pv$ as a solution, as needed. ◀

### 3.2 FBAproFixed

While FBAproFixed admits an expression relying only on basic linear algebra, derived in Appendix Section C, it does not generalize well to the general problem. For this, we rely on a closed-form expression for an orthogonal projection to the intersection of two affine space given in [25, Theorem 4.1], restated here:

▶ **Theorem 6.** *If $T := (a + col(A)) \cap (b + col(B))$ is not empty, and with $D := AA^* + BB^*$, then $T = c + col(C)$, where*

$$c = AA^* D^+ (b - a) + a$$

*and, denoting column concatenation with :,*

$$C = \left[BB^* D^+ A : AA^* D^+ B\right].$$

We can now derive the affine intersection formulation of FBAproFixed:

**Proof of Theorem 5.** Recall that $A$ is an orthogonal basis spanning the steady-state space, and that $B$ is defined such that $\forall x. x + col(B) = \{y | y_H = x_H\}$. Then the space we seek an orthogonal projection to is $col(A) \cap (v + col(B))$. Following Theorem 6, and given $A, B$ are real, we define

$$\begin{aligned} D &= AA^T + BB^T \\ c &= AA^T D^+ v \\ C &= [BB^T D^+ A : AA^T D^+ B], \end{aligned}$$

and the projection of $v$ to $c + col(C) = col(A) \cap (v + col(B))$ is

$$CC^+(v-c)+c = CC^+(v-AA^T D^+ v)+AA^T D^+ v = \left(CC^+(I_r - AA^T D^+) + AA^T D^+\right) v. \blacktriangleleft$$

## 3.3 FBApro

To derive FBApro, we combine the restriction method used for FBAproPartial with the affine intersection representation of FBAproFixed:

**Proof of Theorem 1.** With $c$ and $C$ as before, and with $P := P_M$, we want to find $x$ solving

$$\begin{aligned} \min_{x \in \mathbb{R}^r} \quad & \|x_M - v_M\|_2^2 \\ \text{subject to} \quad & Sx = 0 \\ & x_H = v_H, \end{aligned}$$

equivalently expressed as

$$\min_{x \in c + col(C)} \|v_M - x_M\|_2^2 = \min_{y \in Pc + col(PC)} \|Pv - y\|_2^2,$$

and solved by an orthogonal projection of $Pv$ to $Pc + col(PC)$. Denoting $E = C(PC)^+ P$, this is given by

$$\begin{aligned} & PC(PC)^+(Pv - Pc) + Pc \\ &= PE(v - AA^T D^+ v) + PAA^T D^+ v \\ &= \left(PE(I_r - AA^T D^+) + PAA^T D^+\right) v \\ &= P\left(E(I_r - AA^T D^+) + AA^T D^+\right) v. \end{aligned}$$

Given a projection of $Pv$ to $Pc + col(PC)$ denoted $y$, we now seek $x$ such that $Px = y$. As in FBAproPartial, our expression yields such a solution since it is a right product of $P$, and thus,

$$FBApro(S, M, H, v) = \left(E(I_r - AA^T D^+) + AA^T D^+\right) v. \blacktriangleleft$$

# 4 Use Cases

In this section, we illustrate the use of FBApro and its variants on a toy metabolic network, demonstrating their behavior on data imputation.

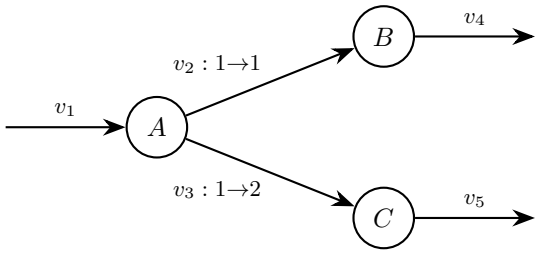



**Figure 2** Toy branched pathway. One unit of $A$ is converted to either one unit of $B$ (via reaction $v_2$) or two units of $C$ (via reaction $v_3$).

## 4.1 Toy Model

Consider the branched pathway in Figure 2, loosely analogous to a metabolic branch point such as the splitting of a precursor between two biosynthetic routes with different stoichiometric yields. The network has three metabolites $A, B, C$, five reactions, and the stoichiometric matrix

$$S = \begin{pmatrix} 1 & -1 & -1 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 2 & 0 & -1 \end{pmatrix},$$

where reaction $v_1$ imports $A$ from the environment, $v_2$ converts one unit of $A$ into one unit of $B$, $v_3$ converts one unit of $A$ into two units of $C$, and $v_4, v_5$ export $B$ and $C$ respectively. The steady-state space is two-dimensional, parametrized by $t, s \in \mathbb{R}$ as

$$ker(S) = \{(t+s,\ t,\ s,\ t,\ 2s) \mid t, s \in \mathbb{R}\}.$$

Note that the network definition does not explicitly include bounds on reactions, or the requirement that $t, s \geq 0$.

## 4.2 Data Imputation on Toy Model

Suppose the ground-truth steady-state flux is $v^* = (7, 4, 3, 4, 6)$, corresponding to $t = 4, s = 3$. We observe a partial and noisy reference: $v_1 = 7$ is measured with high confidence, $v_3 \approx 2.2$ and $v_4 \approx 3.6$ are measured with moderate noise, while $v_2$ and $v_5$ are unobserved. We assign $H = \{1\}$, $M = \{3, 4\}$ for FBApro (FBAproFull henceforth, for readability), and set the reference (input) vector $\tilde{v} = (7, 0, 2.2, 3.6, 0)$ where entries for unobserved reactions are set to zero. For FBAproPartial, we set $M = \{1, 3, 4\}$ and treat the first reaction as a noisy input. For FBAproFixed, we set $H = \{1\}$ and $M = \{2, 3, 4, 5\}$, and for FBAproBasic $M = [r]$. Table 1 shows the output of each FBApro variant.

The results illustrate the value of the three-level confidence structure. FBApro correctly treats the high-confidence measurement as a hard constraint, fits the noisy measurements as well as the steady-state constraint allows, and does not penalize deviations on the unobserved reactions. Each special case loses one of these advantages: FBAproPartial treats the measurement of $v_1$ as noisy, losing information, FBAproFixed assumes unmeasured fluxes should be minimized in absolute value, and FBAproBasic suffers from both effects simultaneously. While this order of performance is expected, we note that different orders are possible depending on different factors, such as the number of unmeasured reactions, or a positive or negative bias in noisy values.

**Table 1** Data imputation results on the toy model. The reference $\tilde{v}$ uses zero for unobserved reactions (marked with "u"). Medium-confidence or approximate values are marked with ≈. Distances are $L_2$ distances of the output $\hat{v}$ to the ground truth $v^*$.

| | $v_1$ | $v_2$ | $v_3$ | $v_4$ | $v_5$ | $\|\hat{v} - v^*\|$ |
|---|---|---|---|---|---|---|
| Ground truth $v^*$ | 7 | 4 | 3 | 4 | 6 | – |
| Reference (input) $\tilde{v}$ | 7 | 0 (u) | ≈ 2.2 | ≈ 3.6 | 0 (u) | – |
| FBAproFull $\hat{v}$ | 7 | 4.2 | 2.8 | 4.2 | 5.6 | 0.53 |
| FBAproPartial $\hat{v}$ | 6.6 | 4 | 2.6 | 4 | 5.2 | 0.98 |
| FBAproFixed $\hat{v}$ | 7 | 5.2 | 1.8 | 5.2 | 3.6 | 3.17 |
| FBAproBasic $\hat{v}$ | 4.2 | 3.2 | 1 | 3.2 | 2 | 5.4 |

While this imputation problem is natural for FBApro, it does not have a natural representation using FBA and most derived methods. FBA can encode noisy observations only as flux bounds (e.g. $v_3 \in [2.5 - \epsilon, 2.5 + \epsilon]$), requiring an explicit noise model and an appropriate choice of $\epsilon$. iMAT reduces the quantitative reference to a ternary classification of reactions as ON, OFF or unconstrained, discarding the numerical signal. MoMA, being similar in formulation, can recreate the performance of FBAproBasic, and analogues of FBAproPartial and FBAproFixed could be formulated as quadratic programs with the appropriate modifications. However, to our knowledge, partial and fixed variants of MoMA have never been introduced, and would regardless require the optimization of a quadratic program (or a relaxed linear program).

# 5 Empirical Validation

## 5.1 Methods and Their Inputs

FBApro and the benchmark methods differ in the inputs they require and in how they use the available reference information. In all experiments, the underlying metabolic model provides a stoichiometric matrix and default reaction bounds. The methods then differ in whether they additionally require an objective function, a reference vector, or reaction confidence sets. All methods other than FBApro and its variants are run via their implementation in cobrapy [8].

**FBApro.** FBApro variants take as input a metabolic model and a reference flux vector, together with reaction confidence information. FBAproBasic uses only the reference flux and the model, without distinguishing among reactions by confidence. FBAproPartial additionally uses a set of medium-confidence reactions, denoted $M$, whose reference values are treated as approximate measurements to be matched as closely as possible. FBAproFixed uses a set of high-confidence reactions, denoted $H$, whose reference values are enforced exactly, while optimizing over the remaining reactions. FBAproFull uses both sets: medium-confidence reactions $M$ and high-confidence reactions $H$.

By running special cases of FBApro on data, we effectively ignore some of the information on reaction confidences. Let $M_0, H_0$ be the confidences we derived from the data. FBAproBasic uses $M = [r]$, ignoring the different confidences completely. FBAproPartial uses $M = M_0 \cup H_0$, interpreting any reaction with associated data as medium confidence, and FBAproFixed uses $H = H_0, M = [r] \setminus H_0$, interpreting all reactions without high confidence as medium confidence.

**FBA.** FBA takes as input a metabolic model, specifically a stoichiometric matrix and reaction bounds, as well as a linear objective function, and computes a steady-state flux vector within the reaction bounds maximizing the objective (see Appendix Section A.1 for details). In all experiments, we used each model's default objective, typically the biomass reaction. To apply FBA in settings with gene expression or flux measurements, we used the reference values to modify the model's reaction bounds, allowing the measurements to constrain the feasible flux space. For a reaction with reference value $x$ in a particular sample, and with an assumed gap $l$, we set the reaction bounds to $[x - l, x + l]$. In the synthetic data experiments, we used $l = 10^{-5}$ for fixed reactions, and $l = |X| + 10^{-5}$ for noisy reactions (matching the model we used to generate noise). For real data, we used $l = |X|$. For unmeasured reactions, we used the model's bounds.

**iMAT.** iMAT takes a metabolic model and a reference vector as input, which it interprets as activity measures derived from gene expression (GE) data. It then finds a steady-state flux vector with high agreement to the reference vector under steady-state constraints (see Appendix Section A.3). As with FBA, we use the reference values to define reaction bounds for each sample.

**MoMA.** MoMA takes as input a metabolic model and a reference flux vector, which it interprets as fluxes, and finds via quadratic programming the feasible closest steady-state flux vector to that reference under steady-state constraints (see Appendix Section A.3). Due to technical issues with the cobrapy implementation of MoMA when using sample-specific bounds, we use each model's default reaction bounds for MoMA in all samples.

## 5.2 Timing Methods

To measure the running time of FBApro and several benchmark methods, we retrieved five metabolic models for representative organisms from the BiGG Models repository [22]: a small *E. coli* core metabolism model (e_coli_core), a genome scale model for *E. coli K-12* (iML1515), a yeast model (iND750), a mouse model (iMM1415) and a human model (RECON1). For each model, we first compute a spanning basis for its steady-state flux space and sample 50 steady-state flux vectors by taking linear combinations of the basis vectors with coefficients uniformly sampled from $[0, 1]$. We then select uniformly at random 10% of the reactions to serve as known reactions, and 50% of the model's reactions as unmeasured reactions.

To generate reference fluxes from the sampled steady-state flux vectors, we retain the values of the known reactions, set the unmeasured reaction values to zero, and add multiplicative noise to the remaining reactions as follows. For a true value $x$, we sample the corresponding noisy value $\tilde{x}$ uniformly from $[x - |x|, x + |x|]$. As described in Subsection 5.1, for methods requiring bounds as inputs, we use the model's default bounds for unknown reactions, $\tilde{x} \pm \epsilon$ with $\epsilon = 10^{-5}$ for known reactions, and for noisy reactions, $[\tilde{x} - |\tilde{x}| - \epsilon, \tilde{x} + |\tilde{x}| + \epsilon]$. Note that these bounds do not necessarily contain the true steady-state fluxes. More generally, deriving bounds from empirical measurements that are both feasible and informative is a challenging problem.

We measure the running time of FBApro and the benchmark methods on these simulated fluxes. For FBApro variants, we separately measure the setup time and application time. The setup time consists of computing the projection matrix used by each method, whereas the application time is the time required to apply this precomputed projection to a new reference flux. This distinction is important because the setup cost is paid once for a fixed

model and measurement structure (i.e., which reaction measurements are fixed, noisy, or unknown), while the projection can then be reused across many samples. We note that while FBA, iMAT and MoMA have associated setup steps, notably constructing the associated linear or mixed-integer program, these are not explicitly exposed through cobrapy. Our measurements for these methods capture the average running time of a single method call. Some of the setup time, specifically the base steady-state linear program, is done by cobrapy at model load time and is thus not captured by our measurements.

For FBApro, we separately measure the setup and running time using CPU and GPU. We also measure the amortized application time per sample when the precomputed projection is applied to all 50 samples as a single input matrix. All time measurements were taken on a node with 1 CPU core, 1 NVIDIA MIG GPU, 32GB of CPU memory and 10 GB of GPU memory, allocated from the Della cluster of Princeton Research Computing.

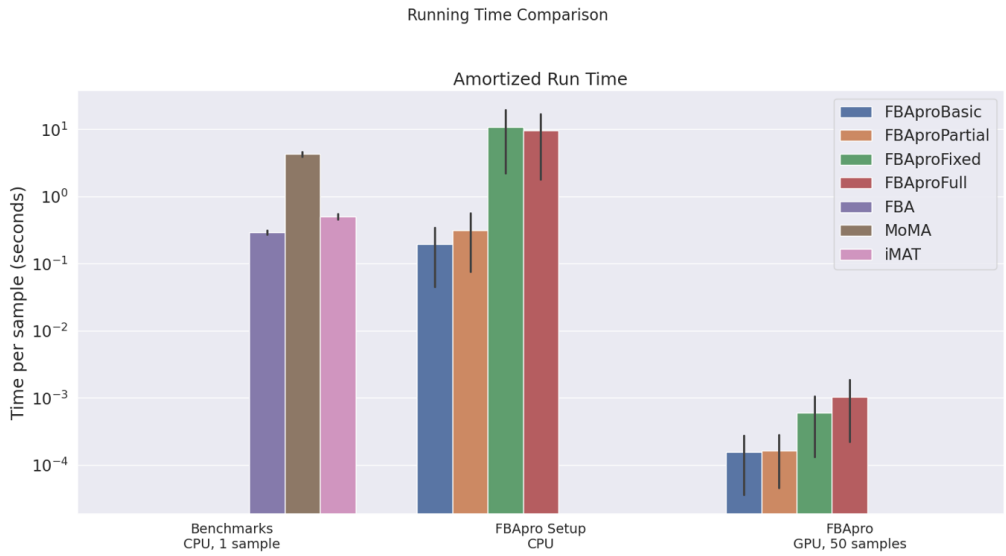



**Figure 3** Timing results, in seconds, on a log scale, averaged over five models and 50 samples per model. For FBApro variants, setup time on CPU is measured separately, and amortized runtime per sample on GPU is measured with the inputs given as a single matrix. FBApro and its variants are orders of magnitude faster than the previous approaches based on FBA, MoMA, and iMAT. Error bars represent a 95% confidence interval.

For reasons relating to the numerical stability of underlying linear algebra drivers and their availability on CPU and GPU, we recommend performing the FBApro setup on CPU, and applying the resulting projection in batches on GPU. Figure 3 shows the corresponding running time results, with the full comparison available in Appendix Figure 6. A single application of any of the FBApro projection methods is orders of magnitude faster than a single application of any of the benchmark methods. This speedup is especially pronounced in the batched setting, where the fixed setup cost is amortized across many reference fluxes.

The main computational consideration is that the setup time for FBApro variants, particularly the Fixed and Full versions, is comparable to the running time of MoMA. However, this is a fixed cost incurred only once for a given metabolic model and measurement configuration, after which each additional reference flux can be processed very quickly. Thus, FBApro is particularly advantageous in settings involving repeated inferences over many samples measured on the same reactions set. These situations naturally arise, for example, when trying to infer fluxes from gene expression data for large numbers of tumor samples

(such as from TCGA [13]); in this case, across samples, the same reactions with known associated genes have estimates of activity that can be used to constrain flux estimates. For the simplest variant, FBAproBasic, the precomputation is dependent only on the metabolic model, and can be done on model load time, as currently implemented in cobrapy for the setup of standard FBA.

## 5.3 Inferring Hidden Synthetic Data Fluxes

We next evaluate the ability of FBApro and its variants to infer hidden fluxes with a similar experimental setup. We generated synthetic data on the human metabolic model RECON1. First, 2% of the reactions are randomly selected as known, a random 85% are treated as unmeasured, and the remaining 13% are provided with a multiplicative noise of 100%. 25 steady-state samples are then generated, and processed to reference flux input vectors. After running each method on these data, for each noisy reaction, we compute the Spearman correlation between the fluxes returned by each method and the corresponding ground truth fluxes across samples. We similarly compute the Spearman correlation across the same reactions for each sample. For any method, if the predictions are constant across the computed axis, we assign a correlation of zero. We repeat the experiment for 10 independent selections of known and unmeasured reactions, and aggregate the results, resulting in correlations over 250 samples in total.

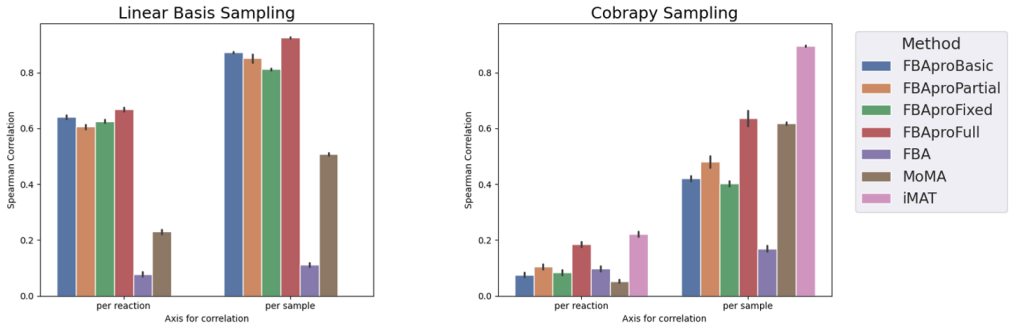
Figure 4a shows the average Spearman correlations for each method, computed both across reactions and samples. In this setting, iMAT returned constant zero predictions. The FBApro variants outperformed the other benchmark methods, with FBAproFull achieving the best correlations on both the reaction and sample axes. All FBApro variants performed better than MoMA and FBA, demonstrating that FBApro can recover hidden flux structure more accurately than existing approaches when the synthetic data are generated from the steady-state space.

To assess the sensitivity of these results to the details of the synthetic data generation procedure, we repeated the experiment using steady-state flux samples generated through cobrapy's sampling method, which produces samples constrained by the model's reaction bounds. These results are shown in Figure 4b. Overall, correlations are substantially lower for all methods under this sampling scheme, particularly when comparing predictions for each reaction across samples. In this setting, iMAT achieves the highest correlations while most FBApro variants continue to outperform FBA, and FBAproFull remains competitive with MoMA.

Taken together, these experiments show that FBApro achieves state-of-the-art flux recovery performance, although the top-performing method can depend on the sampling procedure used to generate synthetic fluxes. Across both sampling schemes, FBApro remains among the stronger performing methods, and in the first setting it is the clear best performer. These accuracy results, together with the amortized timing advantages described above, indicate that FBApro provides a favorable combination of predictive performance and computational efficiency.

## 5.4 Predicting Cancer Cell Line Exchange Rates

To evaluate FBApro on real data, we used metabolite uptake and secretion rates measured for 52 cell-lines from the NCI60 panel [14]. These measurements were matched to 49 existing exchange reactions in the RECON1 metabolic model. We additionally matched 45 of these cell lines with GE data from CCLE [3], reported in TPM units. The GE data was normalized and mapped to the model reactions to produce reaction activities in the range $[0, 1]$. To




**(a)** Spearman correlations between ground truth fluxes and method predictions, data sampled using random combinations of a spanning basis.

**(b)** Spearman correlations between steady-state fluxes and method predictions, data sampled using cobrapy's flux sampling algorithm.

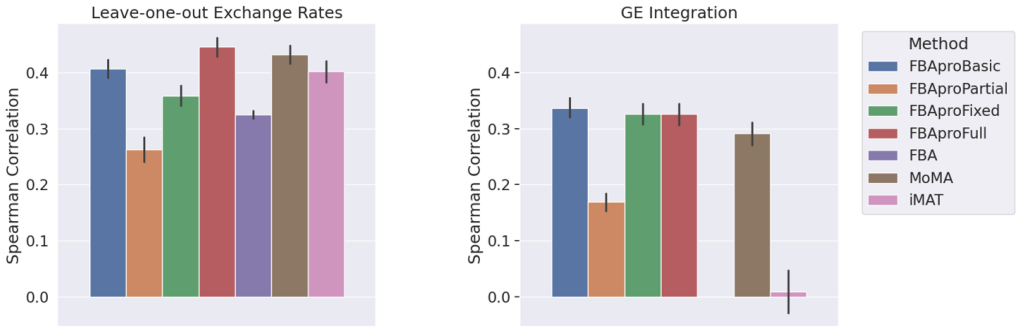
**Figure 4** Method performance on the synthetic data inference experiments. Spearman correlations between method predictions given partially noisy and missing data, measured on noisy reactions for each sample and each reaction. Error bars represent a 95% confidence interval.

obtain reaction-specific flux ranges, we ran FVA on the RECON1 model using 90% optimality and loopless constraints; see Appendix Section A.2 for more details. The reaction activities were then mapped to GE-derived flux estimates via a linear mapping for each reaction from $[0, 1]$ to the range between its lower and upper bound. For reactions with no associated data, we used the mean between the lower and upper bound of the reactions as an estimate. See Appendix Section D for more details on GE preprocessing.

We performed two experiments on the data. In the leave-one-out experiment, we use only the exchange rate measurements as reference flux data, filling the rest with the mean of lower and upper reaction bounds from FVA. For each sample, we mask one exchange reaction at a time, replacing its value with the mean of bounds as well. We then apply all methods to predict its flux value, with the unmasked exchange rates considered as fixed values, the masked as unknown, and the rest as noisy. The concatenated predictions across all masked reactions form the predicted fluxes by each method for each sample. Note that this results in $2548 = 49 \cdot 52$ applications of each method to the data, highlighting the importance of efficient methods for working with metabolic data.

In the GE integration experiment, we used only the GE data for reference fluxes as input, and evaluated how well each method recovered the measured exchange rates. Reactions with associated GE data were treated as measured, and the rest as unknown. This setting is more challenging than the leave-one-out experiment because the input data and evaluation targets come from different molecular modalities, and are linked only indirectly through the metabolic model. For both experiments and methods taking bounds as input, we used a gap of $|x|$ for a reaction with reference value $x$.

Figure 5 shows the Spearman correlations between predictions and measured exchange fluxes, computed separately per sample. In the leave-one-out experiment, where the input and target measurements come from the same data source, units and experimental setup, correlations are typically high. FBApro variants are competitive with the other benchmark methods, with FBAproFull performing the best. In the GE integration experiment, where the GE-derived reaction activity data is fundamentally different than the fluxes, most FBApro variants still achieve high correlations and outperform benchmark methods. In contrast, FBA predicts constant zero fluxes for all evaluated reactions, while iMAT predicts 89% of the fluxes as zero. These results suggest that FBApro can effectively integrate heterogeneous real-world measurements and achieve strong predictive performance.




**(a)** Spearman correlations per-sample between NCI60 exchange rate data and method predictions on masked exchanges.

**(b)** Spearman correlations per-sample between NCI60 exchange rate data and method predictions on GE mapped reference fluxes.

**Figure 5** Method performance on the real data inference experiments. Spearman correlations between method predictions and NCI60 exchange rate measurements, computed per sample. In the leave-one-out experiment, each exchange rate is masked in turn and predicted using the remaining measured exchange rates as input. In the GE integration experiment, matched GE data is used to create a reference flux vector and method predictions are compared with measured exchange rate data. Error bars represent a 95% confidence interval.

## 6 Discussion

In this work we present FBApro, a linear transformation solving the problem of finding a closest steady-state flux to a noisy, partial reference flux vector. We define the optimization problem solved by FBApro, as well as special cases of it, derive the expression for the corresponding linear transformations, and prove these are indeed optimal solutions. We demonstrate the usefulness of FBApro on an example toy model. We further demonstrate that FBApro is computationally scalable to large amounts of data, allowing a qualitative difference in the kinds of data and *in vitro* experiments that can be utilized with it. FBApro also performs on par or better than existing methods in denoising and recovering missing data, as well as on predicting fluxes based on GE.

A major factor of the centrality of FBA in metabolic modeling is the ease with which it can be modified. Alongside its relative scalability, this resulted in a wide variety of methods in the field extending the optimization problem or utilizing FBA as a building block within their algorithm. We believe that FBApro, as another simple, flexible method, can also serve as another basic building block and inspire more diverse future methods.

Even without further modification, FBApro has a variety of potential use-cases: GE data integration, where GE of genes for a reaction are used to make initial estimates of fluxes; noise cancellation and data imputation, where noisy and/or missing flux measurements or estimates are mapped to steady-state fluxes; simulation of knockdowns, where wild-type fluxes are replaced with zeros for knocked down reactions; and even simulating different metabolic objectives, by finding the closest steady-state vector to an indicator vector of the corresponding reactions.

An important consideration in using FBApro is that it does not enforce reaction bounds. This enables a substantially simpler framework, but also discards potentially useful information. In FBA and related methods, reaction bounds are often needed to ensure a finite solution; however, their values are typically unknown and frequently set to default values. In our validations, FBApro performs favorably without bounds. Furthermore, context-specific

estimated bounds can be interpreted as estimated flux values with associated confidence ranges and incorporated as reference fluxes. Nevertheless, FBApro cannot enforce reaction directionality constraints, and this remains an important limitation of the approach.

One motivation for replacing linear programming with a linear transformation is that the resulting closed-form differentiable function can easily fit within gradient-based learning frameworks. FBApro could improve metabolic modeling in these frameworks, either as a concept bottleneck preventing overfitting, or as a mechanism for transfer learning by constraining an inner layer to have a structure interpretable as reaction fluxes. Integrating neural architectures with CBMM modeling assumptions is a promising avenue for future research [10, 2], and by mapping model outputs directly into the steady-state space, FBApro may provide a simple but powerful bridge between flexible neural architectures and mechanistically constrained metabolic models.

## References


1 Rasmus Agren, Sergio Bordel, Adil Mardinoglu, Natapol Pornputtapong, Intawat Nookaew, and Jens Nielsen. Reconstruction of genome-scale active metabolic networks for 69 human cell types and 16 cancer types using init. *PLoS computational biology*, 8(5):e1002518, 2012. doi:10.1371/JOURNAL.PCBI.1002518.

2 Norah Alghamdi, Wennan Chang, Pengtao Dang, Xiaoyu Lu, Changlin Wan, Silpa Gampala, Zhi Huang, Jiashi Wang, Qin Ma, Yong Zang, et al. A graph neural network model to estimate cell-wise metabolic flux using single-cell rna-seq data. *Genome research*, 31(10):1867–1884, 2021.

3 Jordi Barretina, Giordano Caponigro, Nicolas Stransky, Kavitha Venkatesan, Adam A Margolin, Sungjoon Kim, Christopher J Wilson, Joseph Lehár, Gregory V Kryukov, Dmitriy Sonkin, et al. The cancer cell line encyclopedia enables predictive modelling of anticancer drug sensitivity. *Nature*, 483(7391):603–607, 2012.

4 Scott A Becker and Bernhard O Palsson. Context-specific metabolic networks are consistent with experiments. *PLoS computational biology*, 4(5):e1000082, 2008. doi:10.1371/JOURNAL.PCBI.1000082.

5 Ariel Bruner and Mona Singh. Singh-Lab/FBApro. Software, swhId: swh:1:dir:56f800231d0dae5268df2adbab6fd5599d0a376f (visited on 2026-08-12). URL: https://github.com/Singh-Lab/FBApro, doi:10.4230/artifacts.27640.

6 Daniel J Cook and Jens Nielsen. Genome-scale metabolic models applied to human health and disease. *Wiley Interdisciplinary Reviews: Systems Biology and Medicine*, 9(6):e1393, 2017.

7 Markus W Covert, Christophe H Schilling, and Bernhard Palsson. Regulation of gene expression in flux balance models of metabolism. *Journal of theoretical biology*, 213(1):73–88, 2001.

8 Ali Ebrahim, Joshua A Lerman, Bernhard O Palsson, and Daniel R Hyduke. Cobrapy: constraints-based reconstruction and analysis for python. *BMC systems biology*, 7:1–6, 2013.

9 Jeremy S Edwards, Markus Covert, and Bernhard Palsson. Metabolic modelling of microbes: the flux-balance approach. *Environmental microbiology*, 4(3), 2002.

10 Léon Faure, Bastien Mollet, Wolfram Liebermeister, and Jean-Loup Faulon. A neural-mechanistic hybrid approach improving the predictive power of genome-scale metabolic models. *Nature Communications*, 14(1):4669, 2023.

11 David A Fell and J Rankin Small. Fat synthesis in adipose tissue. an examination of stoichiometric constraints. *Biochemical journal*, 238(3):781–786, 1986.

12 Sean R Hackett, Vito RT Zanotelli, Wenxin Xu, Jonathan Goya, Junyoung O Park, David H Perlman, Patrick A Gibney, David Botstein, John D Storey, and Joshua D Rabinowitz. Systems-level analysis of mechanisms regulating yeast metabolic flux. *Science*, 354(6311):aaf2786, 2016.

13 National Cancer Institute. The cancer genome atlast program (tcga). https://www.cancer.gov/tcga. Accessed: 2025-12-11.

14 M. Jain, R. Nilsson, S. Sharma, N. Madhusudhan, T. Kitami, A.L. Souza, and V.K. Mootha. Metabolite profiling identifies a key role for glycine in rapid cancer cell proliferation. *Science*, 336(6084):1040–1044, 2012. doi:10.1126/science.1218595.

**15** Livnat Jerby, Tomer Shlomi, and Eytan Ruppin. Computational reconstruction of tissue-specific metabolic models: application to human liver metabolism. *Molecular systems biology*, 6(1):401, 2010.

**16** John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A. A. Kohl, Andrew J. Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873):583–589, 2021.

**17** Joshua AM Kaste and Yair Shachar-Hill. Accurate flux predictions using tissue-specific gene expression in plant metabolic modeling. *Bioinformatics*, 39(5):btad186, 2023. doi:10.1093/BIOINFORMATICS/BTAD186.

**18** Michael Kohlstedt, Judith Becker, and Christoph Wittmann. Metabolic fluxes and beyond–systems biology understanding and engineering of microbial metabolism. *Applied microbiology and biotechnology*, 88(5):1065–1075, 2010.

**19** Nathan E Lewis, Harish Nagarajan, and Bernhard O Palsson. Constraining the metabolic genotype–phenotype relationship using a phylogeny of in silico methods. *Nature Reviews Microbiology*, 10(4):291–305, 2012.

**20** Daniel Machado and Markus Herrgård. Systematic evaluation of methods for integration of transcriptomic data into constraint-based models of metabolism. *PLoS computational biology*, 10(4):e1003580, 2014. doi:10.1371/JOURNAL.PCBI.1003580.

**21** Miha Moškon and Tadeja Režen. Context-specific genome-scale metabolic modelling and its application to the analysis of covid-19 metabolic signatures. *Metabolites*, 13(1), 2023. doi:10.3390/metabo13010126.

**22** Charles J Norsigian, Neha Pusarla, John Luke McConn, James T Yurkovich, Andreas Dräger, Bernhard O Palsson, and Zachary King. Bigg models 2020: multi-strain genome-scale models and expansion across the phylogenetic tree. *Nucleic Acids Research*, 48(D1):D402–D406, November 2019. doi:10.1093/nar/gkz1054.

**23** Jeffrey D Orth, Ines Thiele, and Bernhard Ø Palsson. What is flux balance analysis? *Nature biotechnology*, 28(3):245–248, 2010.

**24** Eleftherios Terry Papoutsakis and Charles L Meyer. Equations and calculations of product yields and preferred pathways for butanediol and mixed-acid fermentations. *Biotechnology and bioengineering*, 27(1):50–66, 1985.

**25** & Odell P. L. Piziak R. Affine projections. *Computers and Mathematics with Applications*, 2004. doi:10.1016/j.camwa.2004.07.001.

**26** Karthik Raman and Nagasuma Chandra. Flux balance analysis of biological systems: applications and challenges. *Briefings in bioinformatics*, 10(4):435–449, 2009. doi:10.1093/BIB/BBP011.

**27** Daniel Segre, Dennis Vitkup, and George M Church. Analysis of optimality in natural and perturbed metabolic networks. *Proceedings of the national academy of sciences*, 99(23):15112–15117, 2002.

**28** Tomer Shlomi, Omer Berkman, and Eytan Ruppin. Regulatory on/off minimization of metabolic flux changes after genetic perturbations. *Proceedings of the national academy of sciences*, 102(21):7695–7700, 2005.

**29** Tomer Shlomi, Moran N Cabili, Markus J Herrgård, Bernhard Ø Palsson, and Eytan Ruppin. Network-based prediction of human tissue-specific metabolism. *Nature biotechnology*, 26(9):1003–1010, 2008.

**30** Allon Wagner, Chao Wang, Johannes Fessler, David DeTomaso, Julian Avila-Pacheco, James Kaminski, Sarah Zaghouani, Elena Christian, Pratiksha Thakore, Brandon Schellhaass, et al. Metabolic modeling of single th17 cells reveals regulators of autoimmunity. *Cell*, 184(16):4168–4185, 2021.

# A Background

## A.1 Constraint-Based Metabolic Modeling and Flux Balance Analysis

Let $S \in \mathbb{R}^{m \times r}$ be a stoichiometric matrix whose rows correspond to metabolites and columns to reactions. The value of $S_{i,j}$ represents the concentration (or amount) of metabolite $i$ (e.g. in micromolar units $\mu M$) produced (or consumed, if negative) by one unit of flux of reaction $j$, in inverse time unit (e.g. $1/sec$). Given a vector of flux rates $v \in \mathbb{R}^r$, the vector $Sv \in \mathbb{R}^m$ represents the rate of change in each metabolite. The steady-state assumption at the heart of CBMM is then $Sv = 0$.

FBA aims to simulate the fluxes in the cell (or tissue, organism, community) that $S$ represents, under the assumption that metabolic processes aim to maximize a linear objective in terms of reaction fluxes $c \in \mathbb{R}^r$. Typically, $c$ represents growth, by giving a positive coefficient to a reaction representing the consumption of different metabolites for the creation of biomass. The objective value for a flux vector $v$ is then $c^T v$. Since the objective is linear, any $v$ with nonzero objective value can be arbitrarily scaled to achieve any objective value. Therefore, CBMM adds constraints to the fluxes of each reaction preventing arbitrary scaling, and potentially encoding real biological constraints of reaction direction, enzyme abundance or metabolite availability. Let $lb, ub \in \mathbb{R}^r$ be vectors representing the lower and upper bounds for reaction fluxes, respectively. FBA is the linear program solving for an optimal flux vector, that is

$$\begin{aligned} max_v \quad & c^T v \\ \text{subject to} \quad & Sv = 0 \\ & v \leq ub \\ & v \geq lb. \end{aligned}$$

## A.2 Flux Variability Analysis

Flux Variability Analysis (FVA) is a method used to automatically adjust the bounds of reactions in a model to represent their feasible flux ranges. Intuitively, every solution to FBA, for a non-degenerate model, has critical inequalities that are satisfied with an equality in an equation in that solution. However, an inequality may remain unsatisfied in any optimal (or near-optimal) solution, e.g. an upper bound on reactions involved in glycolysis may be higher than the maximal rate of uptake and production of glucose in the model. FVA aims to replace reaction bounds such that each one has a near-optimal solution in which it is critical.

Formally, let $\alpha \in [0, 1]$ be a fraction of optimality, and $c, S, lb, ub$ as in the formulation of FBA, and $x$ be the optimal value of FBA's objective. Then for every reaction $j$, FVA computes

$$\begin{aligned} lb^*_j = min_v \quad & v_j \\ \text{subject to} \quad & Sv = 0 \\ & c^T v \geq \alpha x \\ & v \leq ub \\ & v \geq lb, \end{aligned}$$

and similarly $ub^*_j$ with maximization substituted for minimization. This amounts to two FBA-like linear programs run for each reaction in the model.

A common modification to FVA is to solve the aforementioned program with the added requirement of no flux loops, also known as futile cycles or thermodynamically infeasible cycles. These are typically defined as flux vectors with non-zero fluxes only for mass-preserving reactions, and no exchange or demand reactions.

## A.3 Context-Specific CBMM Methods

Many methods have been developed to simulate the metabolism of systems under a specific context. Such a context can be a gene knockout strain, a specific growth medium, measured fluxes or gene expression. We cover two such methods here. Minimization of Metabolic Adjustment (MoMA) receives as input a metabolic model with stoichiometry $S$ and bounds $lb, ub$, and a reference flux $v$. While the method is generic, the original and still standard implementation is in the context of a knockout strain, where $v$ is taken from a wild-type model, whereas $S$ represents some subset of the original model's reactions. MoMA is then the quadratic program

$$\begin{aligned} min_x \quad & \| v - x \|_2^2 \\ \text{subject to} \quad & Sx = 0 \\ & x \leq ub \\ & x \geq lb. \end{aligned}$$

Integrative Metabolic Analysis Tool (iMAT) is designed to simulate reaction fluxes given GE input. Assume the GE has been mapped to reactions resulting in a reference vector $v$ as in MoMA, albeit with nonnegative values. However, given the inherent noise and loose relationship between genes and reaction fluxes, iMAT interprets the reference vector as indicating only which reactions are likely active and which ones aren't. Let $a, b \in [0, 1]$ be fixed thresholds for interpreting active and inactive indications. Let $x$ be the percentile ranks of the values of $v$, such that $x_i \in [0, 1]$ represents the fraction of reactions with lower or equal values, or $|\{j|v_j \leq v_i\}|/r$. Then the reactions of the models are partitioned to $L = \{i|x_i < a\}$, $H = \{i|x_i > b\}$, $M$ otherwise. Presented verbally and simplified here, iMAT is the mixed-integer program maximizing the number of reactions in $H$ with non-trivial flux, and the number of reactions in $L$ with zero flux, conforming to steady-state assumptions. For our experiments, we used $a = 1/3, b = 2/3$.

# B FBAproBasic Proof

Here we prove Theorem 2, representing $FBAproBasic(S, v)$ as a projection to the space $ker(S) = col(A)$. This is a standard orthogonal projection, and while a reader familiar with projections using pseudoinverses can safely skip this proof, it is our hope that some readers will find it instructive and preliminary to the proofs for the more general variants of FBApro.

**Proof of Theorem 2.** Let $S^+$ be the pseudoinverse of $S$, following the real-value versions of the Moore-Penrose conditions, and computable using SVD:

- $SS^+S = S$
- $S^+SS^+ = S^+$
- $(SS^+)^T = SS^+$
- $(S^+S)^T = S^+S$

We want to show that for every $v \in \mathbb{R}^r$, $Q = I - S^+S$ is such that $Qv$ is the closest vector to $v$ in $ker(S)$. First, observe that

$$SQv = S(I - S^+S)v = (S - SS^+S)v = (S - S)v = 0,$$

and so $Qv \in ker(S)$. To show it minimizes distance to $v$ among all steady-state vectors $x \in ker(S)$, note that

$$\|v - x\|_2^2 = \|((v - Qv) + (Qv - x))\|_2^2 = \|v - Qv\|_2^2 + \|Qv - x\|_2^2 + 2\langle v - Qv, Qv - x\rangle.$$

To show that $\|v - x\|_2^2 \geq \|v - Qv\|_2^2$ it suffices to show that $\langle v - Qv, Qv - x\rangle = 0$. Then

$$\begin{aligned}\langle v - Qv, Qv - x\rangle &= (v - Qv)^T(Qv - x) = v^T(I - Q)^T(Qv - x)\\ &= v^T(S^+S)^T(v - S^+Sv - x) = v^T(S^+S)(v - S^+Sv - x)\\ &= v^T(S^+Sv - S^+SS^+Sv - S^+Sx) = -v^TS^+(Sx) = 0.\end{aligned}$$

To represent FBAproBasic using $A$, first recall that its columns are orthonormal, and therefore $A^TA = I$. It is thus easy to verify that the Moore-Penrose conditions hold with $A^+ = A^T$. Since $col(A) = ker(S)$, $AA^Tv = A(A^Tv) \in col(A) = ker(S)$. For distance minimization, let $x \in ker(S)$, then $\exists y.x = Ay$, and

$$\begin{aligned}\langle v - AA^Tv, AA^Tv - x\rangle &= v^T(I - AA^T)^T(AA^Tv - Ay)\\ &= v^T(I - AA^T)A(A^Tv - y)\\ &= v^T(A - AA^TA)(A^Tv - y) = 0.\end{aligned}$$

◀

## C FBAproFixed Basic Linear Algebra Proof

Recall the use of the pseudoinverse for solving systems of linear equations: if the system of linear equations $Tx = y$ admits a solution, then $x = T^+y$ is one. As a note, if multiple solutions exist, $T^+y$ has minimal norm among them, and if no solutions exist, $T^+y$ minimizes the error $\|Tx - y\|_2^2$, leading to its use in least-square regression problems. While not part of our proofs, this suggests that FBApro and FBAproFixed are expected to return an approximate solution even when an exact one agreeing with $v$ on $H$ may not exist. Given the use of the pseudoinverse for solving linear equations, we can prove Theorem 4:

**Proof of Theorem 4.** Consider the constraints $Sx = 0$ and $x_H = v_H$, and assume a solution exists. Treating $Sx$ as a combination of the columns of $S$, separate them to the columns corresponding to $H$ and $H^c$ and then substitute $v_H$ for $x_H$, yielding

$$\begin{aligned}(S_{[m],H})x_H + (S_{[m],H^c})x_{H^c} &= 0\\ (S_{[m],H^c})x_{H^c} &= -(S_{[m],H})v_H.\end{aligned}$$

Denote $Q_0 \in \mathbb{R}^{|H^c|\times|H|}$ defined as $Q_0 = -(S_{[m],H^c})^+(S_{[m],H})$. Then, if the system has any solutions, a particular solution is admitted by

$$x'_{H^c} = Q_0v_H = -(S_{[m],H^c})^+(S_{[m],H})v_H.$$

Since we are interested in a solution minimizing distance to $v$, we use this expression for a solution to express the affine space defined by the previous system of non-homogeneous equations:

$$U = ker(S_{[m],H^c}) + x'_{H^c}$$

Any vector $x$ with $x_H = v_H$ and $x_{H^c} \in U$ satisfies the original constraints, and has

$$\|v - x\|_2^2 = \|v_{H^c} - x_{H^c}\|_2^2,$$

and so fixing $v_H$ and projecting $v_{H^c}$ to the closest point in $U$ solves Problem 4. For any affine space $T + u$, a vector $t + u$ in the space and a vector $w$, we have that

$$\|w - (t + u)\|_2^2 = \|(w - u) - t\|_2^2$$

and so an orthogonal projection minimizing distance to $T + u$, given an orthogonal projection matrix $Q_T$ to $T$, is of the form $w \to u + Q_T(w - u)$. For the linear space corresponding to $U$, following Theorem 2, the corresponding orthogonal projection is $Q_U \in \mathbb{R}^{|H^c| \times |H^c|}$ with

$$Q_U = I_{|H^c|} - (S_{[m],H^c})^+ S_{[m],H^c}.$$

Note that

$$Q_U Q_0 = -(S_{[m],H^c})^+ S_{[m],H} + (S_{[m],H^c})^+ S_{[m],H^c} (S_{[m],H^c})^+ S_{[m],H} = 0.$$

Define $Q \in \mathbb{R}^{r \times r}$ such that

$$\begin{aligned}
Q_{H,H} &= I_{|H|} \\
Q_{H,H^c} &= 0 \\
Q_{H^c,H} &= -(S_{[m],H^c})^+ (S_{[m],H}) && = Q_0 \\
Q_{H^c,H^c} &= \left(I_{|H^c|} - (S_{[m],H^c})^+ S_{[m],H^c}\right) && = Q_U.
\end{aligned}$$

Then $Qv$ fixes $v_H$ as

$$(Qv)_H = (Q_{H,[r]})v = (Q_{H,H})v_H + (Q_{H,H^c})v_{H^c} = (I_{|H|})v_H = v_H,$$

and projects $v_{H^c}$ to $U$ as

$$\begin{aligned}
(Qv)_{H^c} = (Q_{H^c,[r]})v &= (Q_{H^c,H})v_H + (Q_{H^c,H^c})v_{H^c} = \\
&= Q_0 v_H + Q_U v_{H^c} + 0 = \\
&= x'_{H^c} + Q_U v_{H^c} - (Q_U Q_0) v_H = x'_{H^c} + Q_U(v_{H^c} - x'_{H^c}),
\end{aligned}$$

as needed. ◀

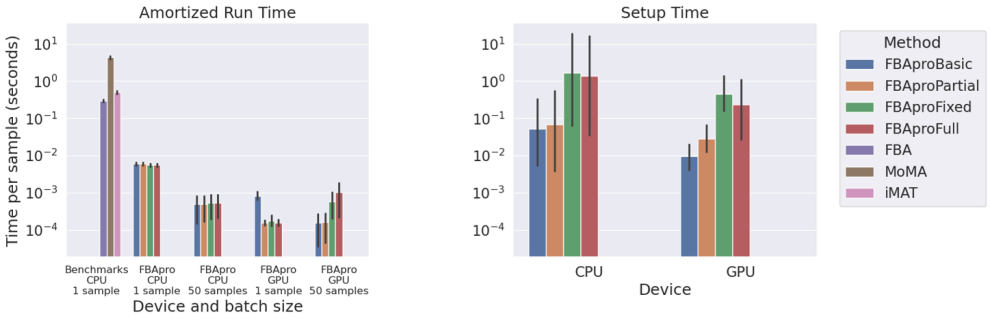
## D Processing and Mapping Gene Expression to Reference Fluxes

GE datasets are processed as follows: Genes with less than 15% non-zero values across samples are dropped. We then apply a quantile transformation to the expression data: denote the number of genes as $g$. We first create a reference distribution by taking, for every rank $i \leq g$, the mean expression value of the $i$-th smallest value in each sample. We then map the values of each sample to this distribution, preserving their ranking. The entire GE matrix is then shifted and scaled so that the values are in the range $[0, 1]$, to represent relative activity.

In order to apply any method to GE data, it must be mapped from the gene space to the reaction space of the model. We follow a typical process for the mapping, although the implementation details are not standardized in the literature [21]. We use the model's Gene-Protein-Reaction (GPR) association, which associates each reaction with a logical expression with gene ids as variables. In the case of multiple transcript variants in the model,

this information is ignored for matching with GE data. The logical operators AND and OR are replaced with geometric mean and arithmetic mean, respectively, and the GE values are substituted for the variables to result in raw reaction activities. Genes not in the data are ignored, so means are taken over the rest. Note that this preserves the range of the GE values.

Let $lb, ub$ denote the lower and upper bounds of a reaction, and $x$ denote its activity in a sample. For a forward reaction, with $lb, ub \geq 0$, the corresponding reference flux is $lb + x(ub - lb)$, such that an activity of 0 corresponds to $lb$ and 1 to $ub$. For backward reactions with $lb, ub \leq 0$, the roles of $lb, ub$ are reversed in the mapping. For reversible reactions with $lb \leq 0 \leq ub$, no value is assigned from activity. These, as well as reactions without a GPR, or without any matching genes in the data, are assigned a reference flux of $(lb + ub)/2$. Note that after running FVA only a small fraction of the reactions remain bidirectional (323/3418).




**(a)** Running time for each method on a single sample, when run on CPU or GPU, and either on 1 sample or amortized over 50 samples. For FBApro, setup time is not included in the running time.

**(b)** Setup time for FBApro variants on CPU and GPU devices.

**Figure 6** Timing results, in seconds, on a log scale, averaged over five models and 50 samples per model. For FBApro variants, runtime is measured on both CPU and GPU, and on the inputs given separately or as a single matrix. Error bars represent a 95% confidence interval.

# Finimap: Fast and Accurate Single-Species Bacterial Pseudoalignment with Finimizers

**Jarno N. Alanko**
Department of Computer Science, University of Helsinki, Finland

**Elena Biagi**
Department of Computer Science, University of Helsinki, Finland

**Simon J. Puglisi**
Department of Computer Science, University of Helsinki, Finland

## Abstract

In recent years, pseudoalignment as a means for mapping reads to databases of reference genomes has become a widely-used method in studies of bacterial pathogenesis. A popular pseudoalignment criterion is thresholded union, in which a read is said to pseudoalign to a reference if the reference contains more than a percentage $t$ of the read's $k$-mers. Several pseudoalignment indexing tools that implement this and other pseudoalignment criteria are now available, including Bifrost, Themisto, and Fulgor. In this paper, we describe a scheme for single-species bacterial pseudoalignment that, instead of $k$-mers, uses shortest unique finimizers (Alanko et al., IEEE/ACM TCBB, 2025) as features for determining pseudoalignment. We show that this scheme, which we call Finimap, leads to a significantly lower false-positive rate than other recent "approximate pseudoalignment" methods Kaminari and Raptor, and is also faster.

**2012 ACM Subject Classification** Applied computing → Bioinformatics; Theory of computation → Data structures design and analysis; Theory of computation → Pattern matching; Theory of computation → Algorithm design techniques; Applied computing → Computational genomics

**Keywords and phrases** Pseudoalignment, sequence alignment, approximate string matching, string processing, $k$-mer, data structures, data compression

**Digital Object Identifier** 10.4230/LIPIcs.WABI.2026.27

**Supplementary Material** *Software (Source Code)*: `https://github.com/ElenaBiagi/Finimap` archived at `swh:1:dir:b8b539b0f246d45853d327fd276ad4cd9efe7f7e`

**Funding** This work was supported by the Research Council of Finland via grant 339070.
*Elena Biagi*: The Finnish Cultural Foundation (grant 00250241) and the Finnish Accademy of Science and Letters (Vilho, Yrjö and Kalle Väisälä Fund).

**Acknowledgements** The authors wish to thank the Finnish Computing Competence Infrastructure (FCCI) for supporting this project with computational and data storage resources.

## 1 Introduction

Pseudoalignment is an approximate form of sequence alignment in which a match between two sequences is quantified as a function of their shared $k$-mer content. Typically, pseudoalignment is performed between a sequence read and an indexed collection of reference genomes. Threshold union is a popular form of pseudoalignment in which all $k$-mers of the query read are searched in the index, and a match is reported if at least a fraction $\tau$ of the matched $k$-mers occurs in a reference genome. Unlike with traditional sequence alignment, the precise genomic coordinates of the match are not reported.

Computationally, pseudoalignment is much cheaper than regular alignment (as computed by tools such as, e.g., minimap [22]) and retains accuracy at a level that is sufficient for many downstream applications. Pseudoalignment originated in RNA-seq quantification [8] as a way to remove a computational bottleneck in aligning reads. It has since found wide


© Jarno N. Alanko, Elena Biagi, and Simon J. Puglisi;
licensed under Creative Commons License CC-BY 4.0

use in the study of bacterial genomics [5, 20, 24, 25] and the problem of metagenomic read assignment [35, 32]. To highlight just two examples: Khawaja et al. [20] use pseudoalignment (in particular the Themisto tool [5]) in the study of diversity and antibiotic resistance in local populations of *E. coli*; and Mäklin et al. [24] based their mGEMS pipeline for genomic epidemiology – a system for tracing transmission of target pathogens – on pseudoalignment. *The single-species bacterial pangenome use case is our focus throughout this paper.*

There now exist several efficient $k$-mer-based pseudoalignment indexing tools, including Themisto [5], Fulgor [9, 10, 12, 30], Metagraph [19], and Bifrost [15]. Recently, there has been a move toward pseudoalignment methods that are not based on $k$-mers which further loosen match criteria with the aim of further reducing query times. These tools can be thought of as approximating the results of $k$-mer based pseudoalignment, even if there are no formal approximation guarantees. Raptor [37] and Kaminari [21], are two such methods that make use of *minimizers* [34, 33, 36] instead of $k$-mers as features. This possibly makes their matches less precise than the $k$-mer based pseudoalignment tools, but their operations faster.

**Contribution.** In this paper, we introduce a new (approximate) method for pseudoalignment based on shortest unique *finimizers* [2], variable length substrings that are unique in the de Bruijn graph of the indexed data set (we give a formal definition below). Our tool, Finimap, indexes and queries only finimizers instead of full $k$-mers, with color sets of $k$-mers sharing the same finimizer merged during indexing. In particular, we present the following results:

1. We demonstrate that Finimap is fast compared to existing approximate methods Raptor and Kaminari, while producing drastically fewer false positive matches relative to established $k$-mer based pseudoalignment tools. For example, when pseudoaligning 50,000 long reads to a set of 3682 *E. coli* genomes, Finimap had a false positive rate of 2% and ran in 30 seconds, compared to 15% by Raptor, which had a similar run time. Kaminari had a false positive rate of 13% and was 2× slower at its fastest setting.
2. In order to explain the superior fidelity of Finimap compared to the minimizer-based Raptor and Kaminari schemes, we elucidate an interesting property of finimizers, namely, that they partition the de Bruijn graph into connected components in a way that minimizers do not. This property may be of independent interest.
3. Since finimizers have variable length, they pose distinct challenges compared to $k$-mers and minimizers (used by Raptor and Kaminari), particularly during the lookup phase. To address this, we propose a novel indexing scheme based on Elias–Fano encoding, that is fast and requires a modest amount of space relative to the total index size.

**Roadmap.** In the next section we set notation and define basic concepts used throughout. Then, in Section 3, we review prior work on indexing for pseudoalignment. Section 4 introduces Finimap, our finimizer-based pseudoalignment index and Section 5 presents experiments measuring its performance. In Section 6 we delve more deeply into properties of finimizers, searching for an explanation for Finimap's superior accuracy compared to minimizer-based pseudoalignment. Conclusions and avenues for future work are then offered.

## 2 Preliminaries

Throughout the paper, a *string* $X[1..n]$ defines a sequence of $|X|$ symbols. Since we deal with bioformatics applications, our alphabet is the DNA alphabet, which consists of four symbols $\Sigma = \{\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{T}\}$. The empty string is denoted $\epsilon$ and $|\epsilon| = 0$. The *substring* of $X$ denoted $X[i..j]$ starts at symbol $i$ and ends at symbol $j$. We also use the half-open interval notation $X[i..j) = X[i..j-1]$ and $X(i..j] = X[i+1..j]$. A *prefix* is a substring starting at position 1 and a *suffix* is a substring ending at position $n$. A *$k$-mer* is a (sub)string of length $k$.

▶ **Definition 1** ($k$-Spectrum). *The $k$-spectrum $S_k(X)$ of a string $X$ is the set of distinct $k$-mers $\{X[i..i+k-1] \mid i = 1, \ldots, |X| - k + 1\}$. The $k$-spectrum $S_k(X_1, \ldots, X_m)$ of a set of strings $X_1, \ldots, X_m$ is the union $\bigcup_{i=1}^{m} S_k(X_i)$*

▶ **Definition 2** (Padded $k$-Spectrum). *Let $R = S_k(X_1, \ldots, X_m)$ be a $k$-spectrum with alphabet $\Sigma$, and let $R' \subseteq R$ be the set of $k$-mers $Y$ such that $Y[1..k-1]$ is not present as a suffix of any $k$-mer in $R$. The padded $k$-spectrum is the set $S_k^+(X_1, \ldots, X_m) = S_k \cup \{\$^k\} \cup \bigcup_{Y \in R'} \{\$^{k-i} Y[1..i] \mid i = 1, \ldots, k-1\}$, with special character $\$ \notin \Sigma$ and $\$ < c$ for all $c \in \Sigma$.*

In a *padded $k$-spectrum* a minimal set of \$-padded *dummy* $k$-mers are added to ensure that in the de Bruijn graph of $S_k(X_1, \ldots, X_m)$ and the dummy $k$-mers, every non-dummy $k$-mer has an incoming path of length at least $k$. Here, we use a padded node-centric definition of a de Bruijn Graph:

▶ **Definition 3** (de Bruijn Graph). *The de Bruijn graph of a set of strings $X_1, \ldots, X_m$ is a directed graph $G = (V, E)$, where each vertex represesents a $k$-mer: $V = S_k^+(X_1, \ldots, X_m)$. And there exists a directed edge from vertex $i$ to vertex $j$, representing $k$-mers $u$ and $v$, repsectuvely, iff $u(1..k] = v[1..k)$.*

The label of an edge $e = (i, j)$, denoted with $\ell(e)$ or $\ell(i, j)$, is the character $v[k]$. The label of a path of edges is the concatenation of the edge labels in the order of the path.

We now define a central notion of this paper: *finimizers*, introduced by Alanko et al. [2].

▶ **Definition 4** (Shortest t-Finimizers). *Let $G$ be the de Bruijn graph of a set of $k$-mers $R$, $X$ be a $k$-mer, and $Y$ be a substring of $X$, and $t \geq 1$ be an integer. We call $Y$ the* shortest $t$-finimizer *of $X$ with respect to the $k$-mer set $R$ if $Y$ occurs at most $t$ times in $G$ – $t$ nodes in $G$ are reached by a path spelling $Y$ – and there is no shorter substring of $X$ with at most $t$ occurrence in $G$.*

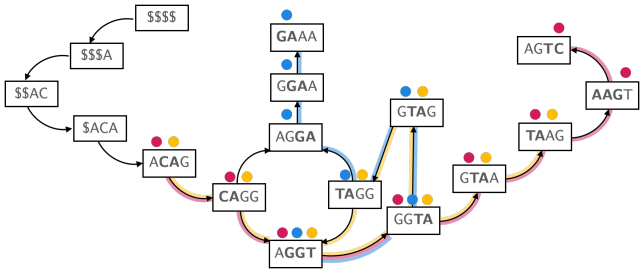
If multiple shortest unique finimizer candidates exist, a consistent tiebreaker such as lexicographic comparison can be used to always pick the same finimizer.

When $t = 1$, the finimizers are called *Shortest Unique Finimizers (SUFs)*. With a careful implementation, a finimizer function for shortest unique finimizers can be constructed in time linear in the total length of the input $k$-mer set [2], and evaluated for every $k$-mer of a query sequence $Q$ in time $O(|Q|)$. Note that a *SUF* exists for every $k$-mer in the index as every $k$-mer appears only once in the dDG. From now on, with a little abuse of notation, when mentioning finimizers we will always refer to *SUF*s. In Figure 1b, finimizers are in bold: the finimizer of AGGT is GGT and not GT as there are two paths on the graph labeled with GT, namely the paths ACAG-AGGT and GTAA-AAGT, but only one path labeled GGT: \$ACA-AGGT.

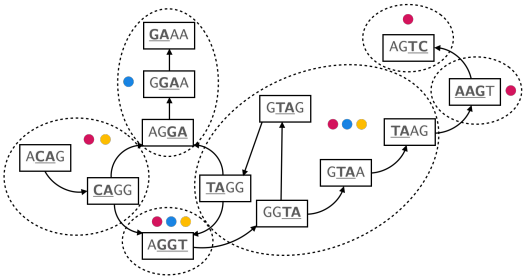
When indexing a set of genomes, we deal with *colored de Bruijn graphs*, where each node represents a $k$-mer from a reference collection, and is labeled with a color indicating the set of references in which the $k$-mer appears (see Figure 1a).

As mentioned in the introduction, there exist several previously published tools [21, 37, 38] that solve pseudoalignment approximately use the concept of minimizers [34, 33, 36]. For completeness we define minimizers now. A minimizer is the smallest $m$-mer of $k$-mer:

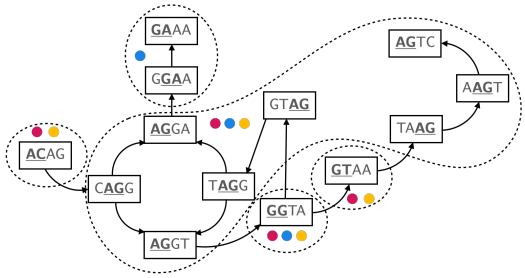
[1] For the sake of this example AA has not been considered a valid lexicographic minimizer as it is usually the case with stretches of As.




**(a)** Colored de Bruijn graph of $\mathcal{T}$.




**(b)** Colored dBG of $\mathcal{T}$ with highlighted finimizer partitions.




**(c)** Colored dBG of $\mathcal{T}$ with highlighted minimizer partitions.

**Figure 1** (a) Colored de Bruijn graph, with $k = 4$, of the set of genomes $\mathcal{T} = \{$ACAGGTAAGTC, ACAGGTAGGTAAG, AGGTAGGAAA$\}$. Nodes are colored based on the genomes in which they appear, their colorset. (b) Color sets for 4-mers sharing the same finimizer ($t = 1$), in bold un underlined, are merged. (c) Color sets for 4-mers sharing the same lexicographic minimizer[1] ($m = 2$), shown in bold and underlined, are merged.

▶ **Definition 5** (Minimizer). *Let $X$ be a $k$-mer, and $m$ be an integer such that $m \leq k$. The minimizer of $X$ is the smallest $m$-mer of $X$ according to a given order relation of the minimizers.*

The chosen order relation could be the simple lexicographic order or, more commonly, something more complex, e.g., based on hashing.

## 3 Pseudoalignment Indexes

Conceptually, indexes for $k$-mer based pseudoalignment associate a color set with each indexed $k$-mer. The color set associated with a given $k$-mer is a list of distinct integer identifiers of sequences in the collection that contain the $k$-mer at least once. This is usually implemented as a dictionary, where the keys are the indexed $k$-mers and the values are *color set ids* that point to the color set associated with the $k$-mer. In the language of information retrieval, this general structure resembles an *inverted index* [31, 42], long the workhorse of web search engines, where the terms are $k$-mers and the postings (document) lists are color sets. This structure was rediscovered in the bioinformatics community over a decade ago as the colored de Bruijn graph [17], and has been much developed since. A survey of many methods can be found in Marchet et al. [26, 27]. Figure 2(a) illustrates a $k$-mer index.

Pseudoalignment tools differ primarily in their choice of data structure for representing the $m$ distinct $k$-mers in the indeded collection and their associated $d$ color sets. Themisto [5] represents the $k$-mer dictionary with the spectral Burrows-Wheeler transform (SBWT) and

the color sets as either arrays of 32-bit integers (for sparse sets) or as bit vectors (for dense sets). Fulgor [10] uses a minimal perfect hash function [29] for the $k$-mer dictionary and has an range of options for the color sets: mFulgor groups genomes into meta-colors removing repetitions across similar genomes, dFulgor encodes each color set relative to a chosen reference, and mdFulgor applies both compression layers together. Other tools aggregate $k$-mers and their relative color set using Bloom Filters organized either in a tree (Sequence Bloom Tree [39] and its derivatives [14, 40, 41]) or in a matrix structure (BGSI [7], COBS [6]).

Loosely speaking, the goal of pseudoalignment is to determine which colors (i.e. indexed sequence) match a given query, with a match determined as some function of shared features (traditionally, $k$-mers) between the query and the sequence. The $k$-mer based approaches mentioned above typically employ a threshold-based pseudoalignment strategy: all $k$-mers of a query sequence are looked up in the index, and a match with sequence $S$ is reported if at least a fraction $\tau$ of the found $k$-mers occur in $S$. A typical value for $\tau$ is 0.8, and $k$ is often 31 when dealing with bacterial genomes. More formally, given a set of documents (e.g. reference genomes) $D_1, D_2, \ldots, D_m$, a query $Q$, and $\tau \in 0..1$, we say $Q$ *matches* or *pseudoaligns* to document $D_i$ if the number of $k$-mers $Q$ that occur in $D_i$ exceeds $\tau$ of the total number of $k$-mers of $Q$ that occur in the set of documents.

| $k$-mer dict. | $\mathcal{C}_k$ |
|---|---|
| AAGT | {●} |
| ACAG | {●, ●} |
| AGGA | {●} |
| ... | ... |
| TAAG | {●, ●} |
| TAGG | {●, ●} |

**(a)** $k$-mer index.

| finimizer dict. | $\mathcal{C}_f$ |
|---|---|
| AAG | {●} |
| CA | {●, ●} |
| GA | {●} |
| GGT | {●, ●, ●} |
| TC | {●} |
| TA | {●, ●, ●} |

**(b)** Finimizer index.

| minimizer dict. | $\mathcal{C}_m$ |
|---|---|
| AC | {●, ●} |
| AG | {●, ●, ●} |
| GA | {●} |
| GG | {●, ●, ●} |
| GT | {●, ●} |

**(c)** Minimizer index.

**Figure 2** Pseudoalignment indexes. In general, a pseudoalignment index for a collection of sequences can be viewed as consisting of a dictionary of features and, for each distinct feature, a set of colors (sequence ids) in which the feature occurs. Three pseudoalignment indexes for the set of genomes $\mathcal{T} = \{$ACAGGTAAGTC, ACAGGTAGGTAAG, AGGTAGGAAA$\}$ are shown above: (a) the $k$-mer-based index with $k = 4$; (b) the finimizer index; (c) the minimizer index with $m = 2$. It is instructive to compare these to the de Bruijn graph pictures in Fig. 1. To pseudoalign a read to an indexed collection, index features appearing in the read are first computed and then the color sets corresponding to these features are aggregated according to a pseudoalignment criterion.

The dictionary/color sets structure followed by $k$-mer indexes generalizes naturally to features other than $k$-mers. The dictionary stores the features in a data structure that supports rapid lookup, and the color sets contain the ids of sequences in the collection that contain those features. Queries are resolved in the same way as for $k$-mers, though the results may now, of course, differ. In Kaminari [21] and Raptor [37] the dictionary contains minimizers and the color set associated with a given minimizer $x$ is the union of the color sets of the $k$-mers in the collection for which $x$ is the minimizer. Kaminari represents the minimizer dictionary as a minimal perfect hash function [29] and the color sets in the same representation that Fulgor [10], mentioned above, uses for $k$-mer color sets. Raptor uses a kind of Bloom filter [28] of the minimizers for its dictionary, and may thus return errors at lookup time. The color sets are represented simply as 32-bit integer lists.

These "approximate" pseudoalignment schemes introduce an additional level of approximation on top of what $k$-mer based pseudoalignment has relative to full sequence alignment. The aim is to improve scalability of indexing while keeping false positives acceptably low.

The result of approximate pseudoalignment is indeed a superset of the true set of documents in which the $k$-mers of $Q$ appear $\mathcal{C}_k$; the challenge is that of making it as close as possible to $\mathcal{C}_k$, while still enabling fast and memory-efficient queries. See Figure 2(b) and Figure 2(c).

In the next section we describe Finimap, a new pseudoalignment index that uses finimizers as features instead of $k$-mers or minimizers.

## 4 Finimap

In this section we describe Finimap, an alternative approach to pseudoalignment based on *finimizers* – shortest unique substrings in the de Bruijn graph, defined earlier. Finimap uses finimizers as features for approximate pseudoalignment, and as an alternative to $k$-mers, which are used by established pseudoalignment tools [5, 12, 15], and minimizers, which have been used recently in some tools [21, 37].

In Finimap, each $k$-mer is associated with a finimizer. Finimizers have three properties that maintain the local structure of the de Bruijn graph:

1. Consecutive $k$-mers tend to share the same finimizer: As a result the number of distinct finimizers is drastically smaller than the number of $k$-mers, enabling a substantial reduction of the index size.
2. Consecutive $k$-mers tend to share the same or very similar color sets: Together with property 1, this implies that when a finimizer is observed, there is a high chance of observing its associated $k$-mers. This helps to limit the number of false positives in $\mathcal{C}_f(x)$ for all $k$-mers $x$, since $\mathcal{C}_f(x)$ is formed from the union of similar sets. And it is in contrast with previously discussed approaches of combining color sets of randomly-selected $k$-mers, e.g. using Bloom filters, which might have substantially different color sets.
3. Disjoint $k$-mers in the de Bruijn graph – i.e. nodes that are not connected – cannot share the same shortest unique finimizer. Such unconnected nodes do not belong to the same sequence and therefore do not share the same color set. Consequently, finimizers color sets are more similar to those of single $k$-mers.

Figure 1 (top) illustrates the colored de Bruijn graph of three genomes together with the partitioning of that graph induced by finimizers (bottom left) and minimizers (bottom right). The figure clearly shows that the two properties explained above are respected. It is also apparent that Property 3 is not respected by minimizers, as $k$-mers not connected in the graph share the same minimizer, and are in the same partition. The practical implications of this will be discussed later.

The whole Finimap index includes: finimizers each associated to a color set id, and color sets – both compressed as explained in the next sections.

### 4.1 Color set representation

Because two or more finimizers could share the same color set (e.g. AAG and TC in Figure 2b), the color sets are deduplicated and each finimizer is assigned a unique color set identifier, that is the index of the color set in the color set representation – described below.

Let $C$ be the total number of colors in the dataset, rows in the matrix, the deduplicated color sets are divided into three classes based on the color set size:

- *Sparse color sets*: A set is classified as sparse if the number of colors in the set is less than 1/16 of all the colors. Present colors are stored a integers and listed one after the other. Ending positions of each set are also stored.

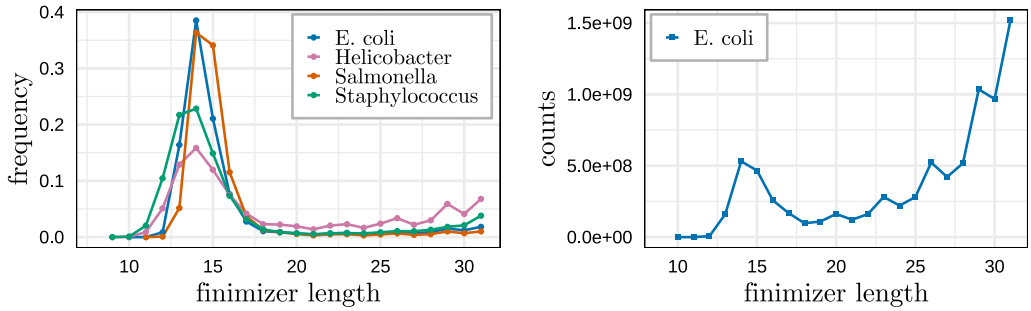



**Figure 3** (Left) Length distribution of the distinct shortest unique finimizers with $k = 31$. The frequency on the $y$-axis is normalized by the total counts of finimizers for each dataset.
(Right) Distribution of finimizer lengths, one finimizer per every 31-mer, in the E. coli index.

- *Dense color sets*: If the set size is in between $[C/16, 15C/16]$, then the set is encoded though a bitvector of $C$ bits. Bits are set only for colors present in the set.
- *Very dense sets*: If the number of colors in the set is higher than $15C/16$, we encode only absent colors using the same encoding used for sparse sets.

The sparse and very dense set are concatenated, and ending positions of each of these sets, monotonically increasing integer sequences, are stored using delta encoding combined with periodic checkpoints sampled every 32 elements. This enables compression while still allowing for constant-time random access. These techniques are inspired by Fulgor [10] and Themisto [5], while being different from both. Figure 9, in Appendix B illustrates the partitioning of the distinct color sets into these three classes across the datasets used in our experiments.

## 4.2 Finimizer dictionary structure

Finimap uses an ad hoc data structure to store and retrieve the finimizers of the indexed collection, which we outline in this section. Our aim was for a dictionary data structure that was fast and that used a reasonable amount of space. We emphasize that this data structure is only a proof-of-concept and is improvable in several ways, both in lookup time and space usage. Future releases of Finimap will explore these improvements, and we sketch what some of them might be throughout this section. Having said this, the current unoptimized version is already pleasingly fast, as our experiments in Section 5 show.

**Finimizer Lookup.** The key operation supported by the dictionary structure when processing a query read $Q$ is to compute, for each $i \in 1..|Q|$, the prefix of $Q[i..|Q|]$, that is a shortest unique finimizer in the dictionary, or, if there is no finimizer, report $\emptyset$.

The dictionary is composed of two parts: $A$, which stores all finimizers having length less than or equal to a user specified parameter $p$, and $B$ which stores the remaining longer finimizers. Let $n_s$ denote the number of short finimizers (i.e. those with length less than or equal to $p$) and $n_g$ denote the number of long finimizers. For ease of exposition we assume the $k$-mer length $k \leq 32$. Recall that $k$ is also an upperbound on the length of any finimizer.

The rationale for splitting the finimizers into short and long sets this way is the observed finimizer length distribution shown in Figure 3 (see also the original paper on finimizers [2]). The division into two sets is a somewhat crude means of allowing us to represent finimizers shorter than $p$ using less bits. A more nuanced approach will be explored in future work.

**Structure for Short Finimizers.** $A$ is essentially an Elias–Fano-like data structure [11, 13] with bucket size $b = 256$, which we now describe. We interpret the short finimizers (i.e., those which are to be stored in $A$) as integers of $2p$ bits. In particular, each short finimizer is padded with 'A' symbols (the symbols are appended) until it is of length $p$. Every symbol is then mapped to a 2-bit integer in the natural way: A $\to$ 00, C $\to$ 01, G $\to$ 10, T $\to$ 11.

For example, with $p = 12$ the finimizer GTACCT of length 6 would become the integer 101100010111000000000000. With $\mathrm{enc}_p(T)$ we denote the binary string of length $2p$ (integer) that results from applying this encoding to the nucleotide string $T$. With $\mathrm{lcp}(i, j)$ we denote the length in bits of the longest common prefix of binary encodings of integers $i$ and $j$.

After encoding the short finimizers as described above, we have a set $S_A$ of $n_s$ integers $a_0 < a_1 < \ldots < a_{n_s-1}$ over a universe $U$, where $|U| = u = 4^p$ – DNA alphabet is of size 4. Observe that the order of the finimizer encodings in $S_A$ corresponds to the lexicographic ordering of the short finimizers, since A<C<G<T corresponds to $00 < 01 < 10 < 11$. Let $E_s = \ell_0, \ell_1, \ldots, \ell_{n_s-1}$ be the sequence of finimizer lengths corresponding to the short finimizers in the order in which their encodings appear in $S_A$. Our lookup procedure for finimizers makes use of the following easily proved lemma.

▶ **Lemma 6.** *Let $x = \mathrm{enc}_p(Q[i..i + k])$ be the integer encoding of the $k$-mer starting at position $i$ in $Q$ and let $a_j = \mathrm{pred}(x, S_A)$ be the predecessor of $x$ in $S_A$. The shortest unique finimizer that is a prefix of $Q[i..i + k]$ is $a_j$ iff $\mathrm{lcp}(x, a_j)/2 \geq \ell_j$.*

The lemma allows us to reduce prefix search on the set of finimizer strings to predecessor search on the set of finimizer encodings. The key observation is that if a short finimizer of length $\ell$ is a prefix of a query string, then their binary encodings share a common prefix of length at least $2\ell$. Hence, after locating the predecessor encoding, we can determine whether it corresponds to a valid finimizer prefix by checking the length of the common prefix.

We will now describe the Elias–Fano-like data structure $A$, used for storing $S_A$, and describe how it supports predecessor queries on $S_A$. A toy example of this is shown in Figure 4.

| short finimizers | padded finimizers | integer ecodings ($S_A$) | bucket | lower 8 bits ($L$) |
|---|---|---|---|---|
| ACGG | ACGGAA | $000110100000 = 416$ | 1 | 160 |
| ACGT | ACGTAA | $000110110000 = 432$ | 1 | 176 |
| AGA | AGAAAA | $001000000000 = 512$ | 2 | 0 |
| CTGAAA | CTGAAA | $011110000000 = 1920$ | 7 | 128 |

**Figure 4** Example of the data structure $A$ used to store short finimizers ($p = 6$), and support predecessor queries. $E = [4, 4, 3, 6]$, $L = [160, 176, 0, 128]$, $B = [0, 1, 1, 0, 0, 0, 0, 1]$, $H = [-, 0, 2, 2, 2, 2, 2, 3]$, buckets 1 start at $L[0]$, bucket 2 starts at $L[2]$, bucket 7 starts at $L[3]$. Note that $B$ and $H$ are not stored separately.

To support efficient predecessor queries, we partition each encoded integer into two parts: its lower 8 bits and its remaining higher bits. The higher bits determine the bucket of the integer (finimizer), while the lower 8 bits are stored explicitly in $L$. Let $L$ be an array of $n_s$ bytes where $L[x]$ is the lower 8 bits of $a_i \in S$. We conceptually divide the universe into buckets of size 256, where the $j$th bucket, denoted $b_j$, corresponds to the interval $[256j, 256(j + 1))$. All integers within the same bucket share the same higher bits and differ only in their lower 8 bits. We store a bitvector $B$ such at $B[j] = 1$ if and only if there is at least one element from $S_A$ falling in bucket $b_j$. We also store an array $H$ of $u/256$ integers, one for each bucket, which contains indexes into $L$, defined as follows. Assume one or more elements from $S_A$ fall into bucket $b_j$ (i.e., $B[j] = 1$) and let $a_i \in S$ be the smallest of those

elements. We set $H[j] = x$ indicating that $L[x]$ is the first entry in $L$ containing lower bits corresponding to elements of $S_A$ falling in bucket $b_j$. If, on the other hand, $b_j$ is empty, then we set $H[j] = y$, such that $a_y$ is the predecessor of $256j$ in $S_A$; in other words, $a_y$ is the largest element in $S_A$ that lies before the start of bucket $b_j$.

To answer query $\text{pred}(x, S_A)$, we let $j = \lfloor x/256 \rfloor$ be the index of the bucket containing $x$. If the bucket containing $x$, i.e., $b_j$, is empty (determined by checking $B[j]$) then the answer to $\text{pred}(x, S_A)$ is given by $H[j]$. Otherwise, if $i_j$'s bucket is nonempty, we calculate the number of elements in it as $c = H[j+1] - H[j] + B[j+1]$ and we scan $L[H[j]..H[j]+c-1]$ until we determine the predecessor of $x$, or, if $x$ is smaller than all elements in the bucket we return $H[j]-1$ as the index of the predecessor. The scan is then limited to the bucket size, making it cache friendly in practice.

When implementing the above scheme we use an array of bytes for $L$ and array of 32-bit integers for $H$. We use the most significant bit of $H[i]$ to store the bit $B[i]$. Queries are very simple, involving just a single access to $H$ followed by a possible scan of at most 256 bytes. The space used is $u/8 + 8n_s$ bits. The array $E$ of finimizer lengths, needed to apply Lemma 6, is stored as a separate array of bytes.

Looking at the example in Figure 4, and given a query 12-mer[2] $Q =$ACGTTAGTGTCA, $\text{enc}_6(Q) = 000110111100 = 444$. We need to find $\text{pred}(444, S_A)$. First we check the bucket: $\lfloor 444/256 \rfloor = 1$. Bucket 1 is not empty ($B[1] = 1$), $H[1] = 0$, $L[0] = 160$ and $\text{lcp}(444, 160)/2 < E[0]$, $3 < 4$, so since $444 > 416$, we need to check the next value in bucket 1. $L[1] = 176$ and $\text{lcp}(444, 176)/2 = E[1] = 4$, thus ACGT is a valid finimizer for $Q$.

There are many ways in which the above structure could be improved. The bucket size of $b = 256$ was chosen the ease implementation as it allows to sequences of lower bits in the EF structure to be stored as an array of bytes. It is almost certainly not the optimal setting for the EF bucket size, which would be $4^p/n_s$. Future work will explore this setting. Also, the array $H$ can be replaced with a bit vector of the bucket sizes encoded in unary, as with usual EF implementations – see, e.g., [23]. Storing $E$ as bytes is also excessive (for $p \leq 16$ 4 bits per entry is sufficient). Also, $E$ could be interleaved with $L$ to reduce cache misses.

**Structure for Long Finimizers.** Let $S_B$ denote the lexicographically ordered set of long finimizers, those with length greater than $p$. The structure for long finimizers, $B$, is essentially a sorted array of 64-bit integers. As with the short finimizers above, we interpret long finimizers as integers. In particular, long finimizers are interpreted as 64-bit integers (recall that we are assuming a maximum $k$, and hence maximum finimizer length, of 32) after padding them to be of length 32 by appending 'A' symbols. In a slight abuse of notation, we will also use $S_B$ to refer to this sorted array of finimizers encoded as integers. We also store an array $L_B$ of finimizer lengths, similar to $L$ in the $A$ structure above.

The same observation about prefix search and predecessor queries used above (i.e., Lemma 6) for short finimizers applies here. That is, to find the shortest long finimizer that is a prefix of $Q[i..i+k]$, we only need to find the predecessor $i_j$ of $Q[i..i+k]$ in $S_B$ and then check whether the longest common prefix of that element and $Q[i..i+k]$ is at least $\ell_j$. Because we store the contents of $S_B$ sorted, answering predecessor queries is easy with binary search. We speed this process up by storing a lookup table of $4^q$ entries one for each possible prefix of length $q$. The $i$th entry of the lookup table contains the subarray of $S_B$ containing all long finimizers prefixed with the $i$th prefix of length $q$ and allows us to jump to the subarray containing finimizers that share that prefix, thus skipping some rounds of binary search.

[2] $k = 12$ and $p = 6$ are just an example.

Much could be done to reduce space in this data structure too. For example, the prefix table removes the need to use to encode the first $q$ symbols of each long finimizer. We also remark that the set of (shortest unique) finimizers is prefix free, and that our current dictionary lookup procedure does not exploit this, for example, to search finimizers in order of length, and terminate search early. We leave these and other optimizations as future work.

## 4.3 Index Construction

The Finimap index is built in two distinct phases: extraction and compression. In the first phase, the finimizers of the dataset are identified and their color sets are constructed. Then finimizers and color sets are compressed, as described in Section 4.2 and Section 4.1, respectively, and each finimizer is assigned a unique identifier corresponding to one of the compressed deduplicated color sets.

**Finimizer identification.** First, we build the SBWT index [4] of the $k$-mers of the input database, and the associated longest common suffix array [1]. Next, we run the streaming finimizer lookup algorithm described in [2] for each sequence in the input database, against the SBWT index of the database. This provides us with the finimizer of each $k$-mer, represented as a pair $(\ell, r)$, where $\ell$ is the length of the finimizer and $r$ is the colexicographic rank of the unique $k$-mer that has the finimizer $\ell$-mer as a suffix. We store the finimizer lengths in an array $F[0..n)$, where $n$ is the length of the SBWT, such that if $k$-mer with colexicographic rank $r$ is suffixed by a finimizer, then $F[r]$ is the length $\ell$ of the finimizer, and otherwise $F[r] = 0$. Computing $F$ is done in parallel with atomic updates on $F$.

**Color set construction.** We construct a bit matrix encoding the color sets of all the finimizers. This is a temporary representation of the color set information, which is eventually deduplicated and stored in the compressed form described above. Let $N$ be the number of identified finimizers, and $C$ be the number of colors. We build a matrix $M[0..N-1][0..C-1]$, such that $M[i][j] = 1$ iff the finimizer with rank $i$ among all the identified finimizers has color $j$. We populate $M$ by querying all input sequences again in parallel against the SBWT index, keeping track of the color of the current input sequence. We find the rank of the finimizer at colexicographic rank $r$ via a rank query on a bit vector that marks the colexicographic ranks of the $k$-mers that are suffixed by finimizers. Color sets are then deduplicated and compressed as explained in Section 4.1.

This color set construction phase could be made more efficient by using Alanko and Puglisi's recent color set construction approach [3] to avoid realising the whole binary matrix.

## 4.4 Pseudoalignment Queries

Given a query read $Q$, Finimap processes each $k$-mer of $Q$ by querying the finimizer dictionary structure to identify matching finimizers and retrieve their associated color sets. These retrieved color sets are then aggregated according to one of the two pseudoalignment criteria described below: *full-intersection* or *threshold-union*. The resulting aggregated color set constitutes the pseudoalignment result for $Q$. Finimap outputs a sorted list of results of the form $(genome, \#\text{found } k\text{-mers})$, ordered by the number of found $k$-mers in decreasing order.

Finimap supports two kinds of queries: *full-intersection* queries and *threshold-union* queries. Given $x$ as a $k$-mer of $Q$ we can define both.

**Full intersection.** A color $c$ is included in the resulting set $R$ if $c$ is present in all the $k$-mers of $Q$ found in the index; that is: $R = \bigcap_{x \in Q} \mathcal{C}_f(x)$.

**Threshold union.** Given a parameter $\tau \in [0, 1]$, a color $c$ is included in the resulting set $R$ if $c$ is present in at least $t = \tau \times |Q|$, where $|Q|$ is the number of the $k$-mers of $Q$ found in the index; that is: $R = \{c \mid |\{x \in Q : c \in \mathcal{C}_f(x)\}| \geq t\}$. Finimap outputs a list of results, sorted by the number of matches.

With $\tau = 1$, full intersection and threshold union queries results are the same. A common value for $\tau$ is 0.8, meaning that at least 80% of the found $k$-mers in $|Q|$ must be present in color $c$ for it to be part of the final result. This threshold is adopted in experiments in Section 5. Figure 5 shows an example of a threshold union query with: color sets of $k$-mers $\mathcal{C}_k$, finimizers $\mathcal{C}_f$, and minimizers $\mathcal{C}_m$, and $\tau = 0.8$.

An important speed up in query time is given by dealing with dense color sets representations by considering only the absent colors and by lowering the number of resulting matches for absent colors and the final threshold $t = \tau \times |Q|$, subtracting the number of dense color sets encountered. This trick was described by the authors of Fulgor [10].

| $Q$ : AGTAAGTC | $C_k$ | | $\mathcal{C}_f$ | | $\mathcal{C}_m$ |
|---|---|---|---|---|---|
| AGTA | $\emptyset$ | AGTA | {●, ●, ●} ✗ | AGTA | {●, ●, ●} ✗ |
| GTAA | {●, ●} | GTAA | {●, ●, ●} ✗ | GTAA | {●, ●} |
| TAAG | {●, ●} | TAAG | {●, ●, ●} ✗ | TAAG | {●, ●, ●} ✗ |
| AAGT | {●} | AAGT | {●} | AAGT | {●, ●, ●} ✗ |
| AGTC | {●} | AGTC | {●} | AGTC | {●, ●, ●} ✗ |
| | {● : 4, ● : 2} | | {● : 5, ● : 3, ● : 3} | | {● : 5, ● : 5, ● : 4} |
| *result* ($\tau = 0.8$): | {● : 4} | | {● : 5} | | {● : 5, ● : 5, ● : 4} |

**Figure 5** Example of a threshold union query, on our running example $\mathcal{T}$, using color sets of $k$-mers $\mathcal{C}_k$, fininimizers $\mathcal{C}_f$ (Finimap), and minimizers $\mathcal{C}_m$.

**False positives.** We compared the results of threshold union queries produced by Finimap (set $\mathcal{A}$, approximate, predicted by our tool) against the results obtained using exact $k$-mer-based pseudoalignment methods (set $\mathcal{X}$, exact), and defined the confusion matrix with respect to the exact pseudoalignment output. True positives (TP) are genomes present in both $\mathcal{A}$ and $\mathcal{X}$, whereas false positives are genomes present only in $\mathcal{A}$, i.e. genomes predicted by Finimap but not reported by the $k$-mer-based pseudoalignment methods.

In Finimap there are two main sources of false positives:

1. The finimizer color set is a superset of color sets of $k$-mers.
2. Alien $k$-mers that are not fully present in the index, but contain a finimizer in the index.

This second source is essentially an exacerbation of the first, arising when a non-indexed $k$-mer inherits the color set of one of the finimizers it contains.

As detailed in the next section, Finimap has fewer false positives than other approximate pseudoalignment tools, as it does not use Bloom filters or hash functions. Like other approximate methods, Finimap presents no false negatives as every $k$-mer in the index is associated to a finimizer.

An example of the result of a threshold union query can be found in Figure 5, where the color sets based on finimizers, $\mathcal{C}_f$, of GTAA and TAAG, differ from the single $k$-mers color sets, introducing false positives, due to $\mathcal{C}_f$ being a superset of $\mathcal{C}_k$ (source 1). AGTA is an alien $k$-mer not considered as such as it contains the finimizer TA (source 2). As shown in the example, false positives are introduced also by approximate methods using minimizers.

**Table 1** Some approximate statistics on the datasets used for experiments. Distinct 31-mers include reverse complement.

| | Ecoli | Salmonella | Staphylococcus | Helicobacter |
|---|---|---|---|---|
| Colors (genomes) | 3,682 | 10,000 | 10,000 | 2,015 |
| Distinct 31-mers ($\times 10^6$) | 341 | 634 | 99 | 185 |
| Distinct finimizers ($\times 10^6$) | 50 | 92 | 16 | 27 |
| Avg. finimizers length | 15.73 | 15.58 | 16.05 | 19.15 |
| Distinct color sets ($\times 10^6$) | 4.82 | 4.56 | 2.04 | 6.73 |
| Avg. genome length (Mb) | 5.15 | 4.85 | 2.80 | 1.63 |

## 5 Experiments

We have implemented our tool FINIMAP in C++. In this section, we report experiments comparing its performance to other pseudoalignment tools. We tested threshold union queries with $k = 31$ and $\tau = 0.8$, on 50,000 positive queries of length 1000.

We compared FINIMAP against two so-called "approximate" pseudoalignment method indexes based on minimizers, KAMINARI [21] and RAPTOR [37], and also to the leading $k$-mer based pseudoalignment index FULGOR [10], which we use as a baseline. Default parameters were employed unless otherwise stated. Only a single thread of execution was used by all tools. See Appendix A for more details.

**Experimental machine.** Experiments were run on a dedicated Linux server running Ubuntu 20.04 with kernel 5.15.0-153-generic. The machine provides 504 GiB of available RAM, and is powered by an AMD Ryzen Threadripper PRO 3975WX processor (32 cores, up to 64 threads, 3.5 GHz clock, 32 × 32 KiB L1 cache, 32 × 512 KiB L2 cache, 128MiB L3 cache). Storage is provided by four 4 TB Seagate ST4000NM000A CMR HDDs, achieving speed of approximately 400 MiB/s, as measured using the `dd` utility.[3]

**Datasets.** In our experiments we consider four **bacterial** datasets that differ by number of genomes, average genome lengths, and internal similarity (repetitiveness).

These included: **E. coli**, a pangenome of 3682 *E. coli* whole genome sequences[4]; **Salmonella**, a collection of 10,000 *S. enterica* genomes[5]; **Staphylococcus**, the first 10,000 *S. aureus* genomes from AllTheBacteria dataset[6] [16]; **Helicobacter**, all 2,015 *H. pylori* whole-genomes in the AllTheBacteria dataset. Some statistics on these data sets are presented in Table 1.

**Results.** We compared the false positive rate (FPR = $\frac{\text{FP}}{(\text{FP}+\text{TN})}$) and query time of our tool to those of other approximate tools for pseudoalignment, and to Fulgor as a baseline – i.e. TP are colors reported by both the approximate tool and Fulgor and FP are colors only reported by the approximate tool.

False-positive rates are shown in Table 2 and the tradeoff between query time and false positive rate for each of the data sets is depicted in Figure 6. FINIMAP is always much faster and more accurate than KAMINARI. When KAMINARI is at its most accurate setting it

[3] Measured using the command `dd if=/dev/zero of=out.bin bs=1MB count=100000 oflag=direct status=progress`.
[4] The collection is available at `zenodo.org/record/6577997`.
[5] First 10,000 files from `http://ftp.ebi.ac.uk/pub/databases/ENA2018-bacteria-661k/`.
[6] Available at `https://allthebacteria.org`.

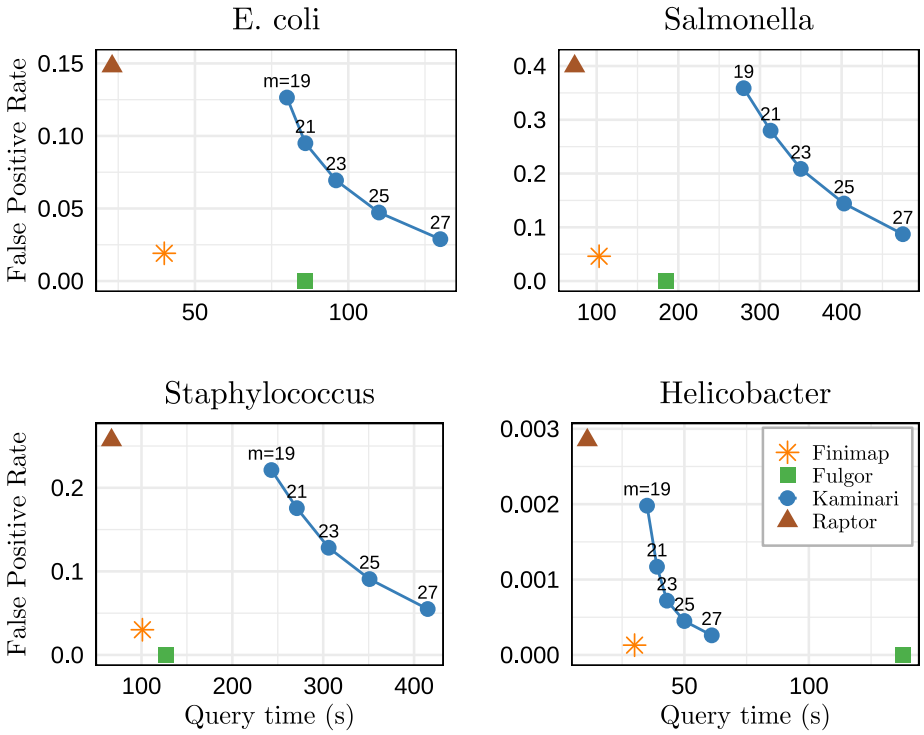



**Figure 6** Results on threshold union queries with 50,000 positive queries of length 1,000, using 1 thread, $k = 31$ and $\tau = 0.8$. Fulgor represents the fastest version of the tool for each dataset: mdFulgor, Fulgor, dFulgor and mdFulgor. Note that Fulgor does not output a sorted list of colors and does not report the number of matches.

produces more false positives than Finimap, but is 2-5× slower. Finimap has comparable speed to Raptor and is always much more accurate, producing roughly a tenth of the false positives that Raptor does. In addition, Finimap is faster than Fulgor, the fastest $k$-mer based pseudoalignment tool. Full results on query time are in the Appendix B, Table 4.

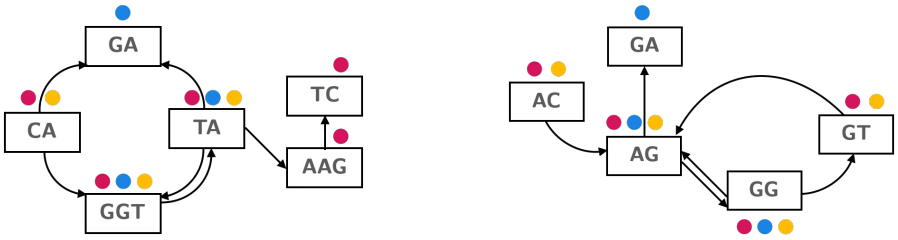
An important performance dimension that is not shown in the plots is that of index size. While it always fits comfortably in RAM, Finimap is always one of the larger indexes tested, about 3-6× bigger than Kaminari at its most accurate setting, with the lookup structure dominating. Finimap is smaller than Raptor, however. Full space results are in Table 3 (Appendix B). We are aware of several places in which the space usage of the current Finimap prototype can be reduced, and leave optimizing index size as future work.

## 6 $k$-merscolor sets vs. *inimizer color sets

In this section we compare $k$-mer color sets and color sets defined by finimizers and lexicographic minimizers. From our experimental results in the previous section, we observe that Finimap has a lower number of false positives compared to the minimizer-based tools Raptor and Kaminari (see Table 2). Here we explore in more detail the differences between the two general approaches, finimizers and minimizers, that could explain the observed variation in the false positive rate.

**Table 2** False positive rate and percentage of true negatives of threshold union queries with 50,000 positive queries of length 1,000, $k = 31$ and $\tau = 0.8$. Color code is blue: $< 2 \times best$, orange: $< 4 \times best$, magenta otherwise, where *best* is the smallest value in every column.

| | | Ecoli | Salmonella | Staphylococcus | Helicobacter |
|---|---|---|---|---|---|
| avg. FPR | Finimap | 0.02 | 0.05 | 0.03 | 0.0001 |
| | Raptor 19 | 0.15 | 0.40 | 0.26 | 0.0029 |
| | Kaminari 19 | 0.13 | 0.36 | 0.22 | 0.0020 |
| | Kaminari 21 | 0.10 | 0.28 | 0.18 | 0.0012 |
| | Kaminari 23 | 0.07 | 0.21 | 0.13 | 0.0007 |
| | Kaminari 25 | 0.05 | 0.14 | 0.09 | 0.0005 |
| | Kaminari 27 | 0.03 | 0.09 | 0.05 | 0.0003 |
| avg. % TN | | 77 | 59 | 60 | 98 |




**(a)** Finimizer-partitioned colored dBG of $\mathcal{T}$. **(b)** Minimizer-partitioned colored dBG of $\mathcal{T}$.

**Figure 7** Finimizer- ((a) $t = 1$) and minimizer-partitioned ((b) $m = 2$) dBG of $\mathcal{T}$. The full dBG of $\mathcal{T}$ is shown in Figure 1.

## 6.1 De Bruijn graph partitions

As illustrated in Figure 1b, consecutive $k$-mers tend to share the same finimizer (Property 1), so that only adjacent $k$-mers in the de Bruijn graph are grouped together (Property 3). This is consistent with the well-known observation that neighboring $k$-mers in the de Bruijn graph generally exhibit identical or highly similar color sets (Property 2).

In contrast, although consecutive $k$-mers tend to share the same minimizer, the partition that minimizers induce on the de Bruijn graph does *not* result in only neighboring $k$-mers sharing the same partition (see Figure 1c). As a result, distant regions of the graph may be included in the same partition, unlike shortest unique finimizers, which preserve locality. Consequently, structural relationships of the original graph are kept in the finimizer minor as only connected compnents are grouped in the same partition (Figure 7a). They are not maintained in the minimizer-partitioned colored de Bruijn graph (Figure 7b), which does not induce a minor: in this example, the edge connecting GT to AG did not exist in the original de Bruijn graph. This suggests that minimizer-based partitioning can distort the underlying graph topology, leading to an increased number of false positives.

## 6.2 Jaccard index

To assess whether the different ways in which the de Bruijn graph is partitioned, described above, have a real impact on the resulting index and queries, we calculated a measure of similarity between the $k$-mer color sets $\mathcal{C}_k$ ($k = 31$), and $\mathcal{C}_m$ or $\mathcal{C}_f$, the color sets of minimizers or finimizers, respectively. We used the Jaccard index [18]:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{\text{TP}}{\text{TP} + \text{FP} + \text{FN}}$$

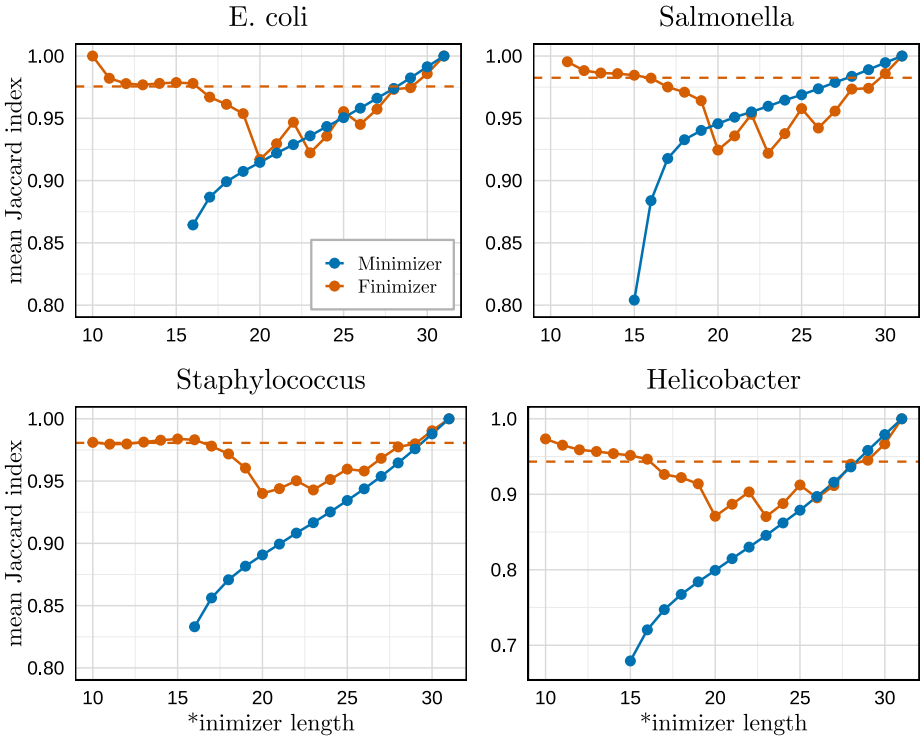



**Figure 8** Plots of the average Jaccard index, the higher the better, of 31-mer color sets and minimizer color sets, in blue, and finimizer color sets, in orange. Since the length cannot be fixed for finimizers, the dashed line represents the weighted mean of the values for all finimizer color sets.

where $A$ corresponds to $\mathcal{C}_k$, and $B$ is either $\mathcal{C}_m$ or $\mathcal{C}_f$. This value was calculated considering all $k$-mer in the datasets with multiplicity 1 or higher. Note that in the formula, FN is always 0. Values range from 0 to 1, where 1 indicates perfectly identical sets, and 0 indicates that the sets are disjoint. Results on four different datasets are shown in Figure 8 for $k = 31$. We calculated the Jaccard index for every $k$-mer in the dataset; $m$ from 15 to 31 for minimizers and $t = 1$ for finimizer. Since finimizers are not of fixed length, we calculated and plotted a weighted mean of the Jaccard index based on the frequency of each finimizer length.

On the E. coli data set with $m = 19$ (default parameter for Kaminari), the average Jaccard index of $\mathcal{C}_k$ and $\mathcal{C}_m$ is 0.9, but with finimizers it is 0.98. In order to achieve a similar value with minimizers we would require $m = 28$, which would clearly have a negative impact on query time, as shown by the trend plotted for Kaminari in Figure 6. For example, with E. coli the value of the mean Jaccard index is lower than the mean Jaccard index for every finimizer length. This suggests that on average, with $m = 19$, finimizer color sets $\mathcal{C}_f$ are more similar to $\mathcal{C}_k$ than minimizer color sets.

## 7 Conclusion and future work

We have described Finimap, a pseudoalignment index that uses finimizers, maximal unique substrings in the de Bruijn graph of the indexed data set, as features, rather than $k$-mers. Our experiments show that the results of Finimap are very similar to $k$-mer based pseudoalignment in that it produces far fewer false positives relative to $k$-mer based indexes

than pseudoalignment tools that use minimizers as features. We have focused on single-species bacterial genomics, which is a significant existing use case for pseudoalignment. Adapting and evaluating FINIMAP for pseudoalignment on broader, possibly more complex data sets, is an interesting direction for future work.

Our current implementation of FINIMAP is reasonably fast: it is always faster than the fastest $k$-mer based index FULGOR in our experiments, often significantly so. It also dominates the minimizer-based index KAMINARI, both in terms of speed and accuracy, and has comparable speed to RAPTOR while producing drastically fewer false positives. We believe our current implementation of FINIMAP can be further optimized, particularly in the lookup phase, in which an ad hoc two-part data structure is currently used. Trie data structures and hashtables stratified on key-length will be explored in future work. More generally, parallelism is a currently unexplored direction for optimization.

In addition, we remark that pseudoalignment indexes such as FINIMAP and KAMINARI get a head start on the $k$-mer based tools they try to emulate, like THEMISTO or FULGOR, by essentially storing some color sets pre-merged, so that less color sets are touched at query time. It may be interesting to try to quantify the computation saved this way and to try to arrive at upperbounds on the speed gains achievable by these new methods – ultimately, it may be possible to accelerate exact tools by caching partial results over a batch of queries.

Finally we are aware of the limitations of pseudoalignment methods that we used as ground truth for our experiments, and believe additional validation with simulated reads with a known ground truth or alignment-based methods such as Minimap2 [22] would provide stronger biological evidence for the accuracy of the proposed method.

## References


1 Jarno N. Alanko, Elena Biagi, and Simon J. Puglisi. Longest common prefix arrays for succinct k-spectra. In Franco Maria Nardini, Nadia Pisanti, and Rossano Venturini, editors, *String Processing and Information Retrieval*, LNCS 14240, pages 1–13. Springer, 2023. doi:10.1007/978-3-031-43980-3_1.

2 Jarno N. Alanko, Elena Biagi, and Simon J. Puglisi. Finimizers: Variable-length bounded-frequency minimizers for $k$-mer sets. *IEEE Transactions on Computational Biology and Bioinformatics*, 22(2):899–910, 2025. doi:10.1109/TCBBIO.2025.3545285.

3 Jarno N. Alanko and Simon J. Puglisi. Construction of distinct $k$-mer color sets via set fingerprinting. *bioRxiv*, 2026. doi:10.64898/2026.02.16.706153.

4 Jarno N. Alanko, Simon J. Puglisi, and Jaakko Vuohtoniemi. Small searchable k-spectra via subset rank queries on the spectral Burrows-Wheeler transform. In *Proc. ACDA*, pages 225–236. SIAM, 2023. doi:10.1137/1.9781611977714.20.

5 Jarno N. Alanko, Jaakko Vuohtoniemi, Tommi Mäklin, and Simon J. Puglisi. Themisto: a scalable colored $k$-mer index for sensitive pseudoalignment against hundreds of thousands of bacterial genomes. *Bioinformatics*, 39(Suppl 1):i260–i269, June 2023. doi:10.1093/bioinformatics/btad233.

6 Timo Bingmann, Phelim Bradley, Florian Gauger, and Zamin Iqbal. COBS: a compact bit-sliced signature index. In *International Symposium on String Processing and Information Retrieval*, pages 285–303. Springer, 2019. doi:10.1007/978-3-030-32686-9_21.

7 Phelim Bradley, Henk den Bakker, Eduardo Rocha, Gil McVean, and Zamin Iqbal. Ultrafast search of all deposited bacterial and viral genomic data. *Nature Biotechnology*, 37:152–159, February 2019. doi:10.1038/s41587-018-0010-1.

8 Nicolas Bray, Harold Pimentel, Páll Melsted, and Lior Pachter. Near-optimal probabilistic rna-seq quantification. *Nature Biotechnology*, 34:525–527, April 2016. doi:10.1038/nbt.3519.

**9** Alessio Campanelli, Giulio Ermanno Pibiri, Jason Fan, and Robert Patro. Where the patterns are: Repetition-aware compression for colored de bruijn graphs. *Journal of computational biology : a journal of computational molecular cell biology*, 31, October 2024. doi:10.1089/cmb.2024.0714.

**10** Alessio Campanelli, Giulio Ermanno Pibiri, and Rob Patro. Fast Pseudoalignment Queries on Compressed Colored de Bruijn Graphs. In Broňa Brejová and Rob Patro, editors, *25th International Conference on Algorithms for Bioinformatics (WABI 2025)*, volume 344 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:21, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.WABI.2025.6.

**11** Peter Elias. Efficient storage and retrieval by content and address of static files. *J. ACM*, 21(2):246–260, April 1974. doi:10.1145/321812.321820.

**12** Jason Fan, Jamshed Khan, Noor Singh, Giulio Ermanno Pibiri, and Rob Patro. Fulgor: a fast and compact $k$-mer index for large-scale matching and color queries. *Algorithms for Molecular Biology*, 19, January 2024. doi:10.1186/s13015-024-00251-9.

**13** R.M. Fano. *On the Number of Bits Required to Implement an Associative Memory*. Computation Structures Group. MIT, 1971. URL: https://books.google.fi/books?id=07DeGwAACAAJ.

**14** Robert Harris and Paul Medvedev. Improved representation of sequence bloom trees. *Bioinformatics (Oxford, England)*, 36, August 2019. doi:10.1093/bioinformatics/btz662.

**15** Guillaume Holley and Páll Melsted. Bifrost: highly parallel construction and indexing of colored and compacted de bruijn graphs. *Genome biology*, 21:249, September 2020. doi:10.1186/s13059-020-02135-8.

**16** Martin Hunt, Leandro Lima, Daniel Anderson, George Bouras, Michael Hall, Jane Hawkey, Oliver Schwengers, Wei Shen, John A. Lees, and Zamin Iqbal. Allthebacteria – all bacterial genomes assembled, available, and searchable. *bioRxiv*, 2025. doi:10.1101/2024.03.08.584059.

**17** Zamin Iqbal, Isaac Turner Mario Caccamo, Paul Flicek, and Gil McVean. De novo assembly and genotyping of variants using colored de Bruijn graphs. *Nature Genetics*, 44:226–232, 2012. doi:10.1038/ng.1028.

**18** Paul Jaccard. Etude de la distribution florale dans une portion des alpes et du jura. *Bulletin de la Societe Vaudoise des Sciences Naturelles*, 37:547–579, January 1901. doi:10.5169/seals-266450.

**19** Mikhail Karasikov, Harun Mustafa, Daniel Danciu, Marc Zimmermann, Christopher Barber, Gunnar Rätsch, and André Kahles. Indexing all life's known biological sequences. *bioRxiv*, 2024. doi:10.1101/2020.10.01.322164.

**20** Tamim Khawaja, Tommi Mäklin, Teemu Kallonen, Rebecca A. Gladstone, Anna K. Pöntinen, Sointu Mero, Harry A. Thorpe, Ørjan Samuelsen, Julian Parkhill, Mateen Izhar, M. Waheed Akhtar, Jukka Corander, and Anu Kantele. Deep sequencing of escherichia coli exposes colonisation diversity and impact of antibiotics in punjab, pakistan. *Nat. Commun.*, 15(5196), 2024. doi:10.1038/s41467-024-49591-5.

**21** Victor Levallois, Yoshihiro Shibuya, Bertrand Le Gal, Yoann Dufresne, Rob Patro, Pierre Peterlongo, and Giulio Ermanno Pibiri. Kaminari: a frugal colored index for approximate k-mer queries. *Bioinformatics Advances*, 6(1):vbag120, January 2026. doi:10.1093/bioadv/vbag120.

**22** Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18):3094–3100, September 2018. doi:10.1093/bioinformatics/bty191.

**23** Danyang Ma, Simon J. Puglisi, Rajeev Raman, and Bella Zhukova. On Elias-Fano for rank queries in FM-indexes. In *Proc. 31st Data Compression Conference (DCC)*, pages 223–232. IEEE, 2021. doi:10.1109/DCC50243.2021.00030.

**24** Tommi Mäklin, Teemu Kallonen, Jarno Alanko, Ørjan Samuelsen, Kristin Hegstad, Veli Mäkinen, Jukka Corander, Eva Heinz, and Antti Honkela. Bacterial genomic epidemiology with mixed samples. *Microbial genomics*, 7(11), 2021. doi:10.1099/mgen.0.000691.

25 Tommi Mäklin, Teemu Kallonen, Sophia David, Christine J Boinett, Ben Pascoe, Guillaume Méric, David M Aanensen, Edward J Feil, Stephen Baker, Julian Parkhill, et al. High-resolution sweep metagenomics using fast probabilistic inference. *Wellcome open research*, 5, 2020. doi:10.12688/wellcomeopenres.15639.2.

26 Camille Marchet. Advancements in colored $k$-mer sets: essentials for the curious. *arXiv preprint*, September 2024. doi:10.48550/arXiv.2409.05214.

27 Camille Marchet, Christina Boucher, Simon Puglisi, Paul Medvedev, Mikaël Salson, and Rayan Chikhi. Data structures based on k -mers for querying large collections of sequencing data sets. *Genome Research*, 31, December 2020. doi:10.1101/gr.260604.119.

28 Svenja Mehringer, Enrico Seiler, Felix Droop, Mitra Darvish, René Rahn, Martin Vingron, and Reinert Knut. Hierarchical interleaved bloom filter: enabling ultrafast, approximate sequence queries. *Genome Biology*, 24, May 2023. doi:10.1186/s13059-023-02971-4.

29 Giulio Ermanno Pibiri. Sparse and skew hashing of $k$-mers. *Bioinformatics*, 38(Suppl 1):i185–i194, June 2022. doi:10.1093/bioinformatics/btac245.

30 Giulio Ermanno Pibiri, Jason Fan, and Rob Patro. Meta-colored compacted de bruijn graphs. In Jian Ma, editor, *Research in Computational Molecular Biology*, pages 131–146. Springer Nature Switzerland, 2024. doi:10.1007/978-1-0716-3989-4_9.

31 Giulio Ermanno Pibiri and Rossano Venturini. Techniques for inverted index compression. *ACM Comput. Surv.*, 53(6), December 2020. doi:10.1145/3415148.

32 Mark Reppell and John Novembre. Using pseudoalignment and base quality to accurately quantify microbial community composition. *PLoS computational biology*, 14(4):e1006096, 2018. doi:10.1371/journal.pcbi.1006096.

33 Michael Roberts, Wayne Hayes, Brian R Hunt, Stephen M Mount, and James A Yorke. Reducing storage requirements for biological sequence comparison. *Bioinformatics*, 20(18):3363–3369, 2004. doi:10.1093/bioinformatics/bth408.

34 Michael Roberts, Brian R. Hunt, James A. Yorke, Randall A. Bolanos, and Arthur L. Delcher. A preprocessor for shotgun assembly of large genomes. *J. Computational Biology*, 11(4):734–752, 2004. doi:10.1089/cmb.2004.11.734.

35 Lorian Schaeffer, Harold Pimentel, Nicolas Bray, Páll Melsted, and Lior Pachter. Pseudoalignment for metagenomic read assignment. *Bioinformatics*, 33(14):2082–2088, 2017. doi:10.1093/bioinformatics/btx106.

36 Saul Schleimer, Daniel Wilkerson, and Alex Aiken. Winnowing: Local algorithms for document fingerprinting. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 10, April 2003. doi:10.1145/872757.872770.

37 Enrico Seiler, Svenja Mehringer, Mitra Darvish, Etienne Turc, and Reinert Knut. Raptor: A fast and space-efficient pre-filter for querying very large collections of nucleotide sequences. *iScience*, 24:102782, June 2021. doi:10.1016/j.isci.2021.102782.

38 Sergey Shiryev and Richa Agarwala. Indexing and searching petabase-scale nucleotide resources. *Nature Methods*, 21:1–9, May 2024. doi:10.1038/s41592-024-02280-z.

39 Brad Solomon and Carl Kingsford. Fast search of thousands of short-read sequencing experiments. *Nature Biotechnology*, 34(3):300, 2016. doi:10.1038/nbt.3442.

40 Brad Solomon and Carl Kingsford. Improved search of large transcriptomic sequencing databases using split sequence bloom trees. In *21st Annual International Conference on Research in Computational Molecular Biology - Volume 10229*, RECOMB 2017, pages 257–271, Berlin, Heidelberg, 2017. Springer-Verlag. doi:10.1007/978-3-319-56970-3_16.

41 Sanjay Kumar Srikakulam, Sebastian Keller, Fawaz Dabbaghie, Robert Bals, and Olga Kalinina. Metaprofi: an ultrafast chunked bloom filter for storing and querying protein and nucleotide sequence data for accurate identification of functionally relevant genetic variants. *Bioinformatics*, 39, February 2023. doi:10.1093/bioinformatics/btad101.

42 Justin Zobel and Alistair Moffat. Inverted files for text search engines. *ACM Comput. Surv.*, 38(2):6, 2006. doi:10.1145/1132956.1132959.

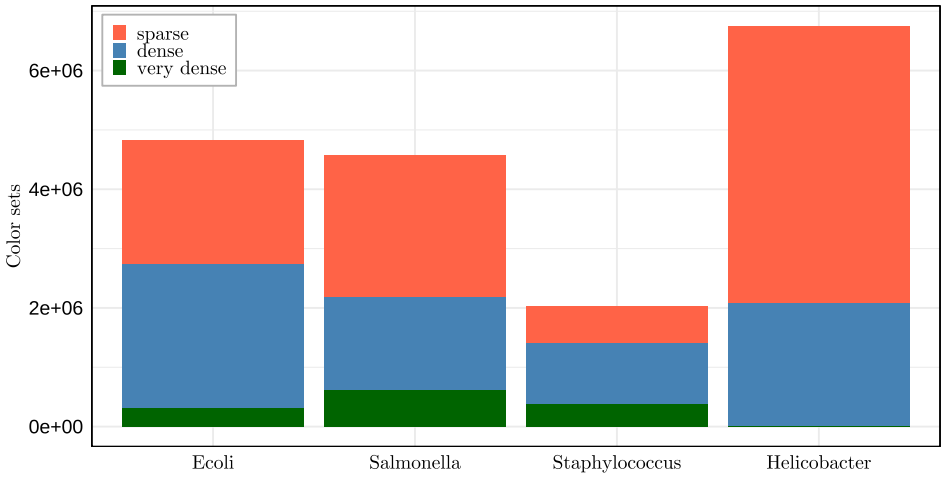



**Figure 9** Finimap color sets representations in terms of number of color sets for each of the three representations.

## A Tool versions and parameters used

- Finimap: parameters `build-fmin -k 31 -x 18 -p 10`
- Fulgor: commit `ef65419`, parameter `-k 31 -m 19`
- Kaminari: commit `0c403d1`, parameters `-k 31 -a -m 19,21,23,25,27`
- Raptor: commit `fa80e93`, parameters `-kmer 19 -window 31`

A threshold $\tau$ of 0.8 was used in all the tools for threshold union queries.

## B Additional data

This section contains some additional results that where not included in the main body.

In Figure 9 it is shown for each dataset the partition of color sets in the three representations: sparse, dense and very dense. The dense representation is the most used in Ecoli and Staphylococcus, whereas in Salmonella and Helicobacter there are more sparse sets. In Helicobacter the sparse representation dominates the other two, and very few colorsets are stored as very dense.

Figure 10 shows FINIMAP index size split into colors and finimizers representations. Distinct dataset have different percentage of the index size dedicated to the two representations. For example, Helicobacter only contains 2015 genomes, but in spite of these genomes being the shortest out of the four used datasets, their high level of repetitiveness leads to 27 millions of finimizers quite long on average (19.15 bases) and this also creates a high number of distinct color sets. Nevertheless the size of color sets representations is smaller than that of other datasets due to the lower number of genomes; still the finimizers representation dominates. On the contrary, in Salmonella (10,000 genomes) the color sets representation dominates the index size with 4,564,828 distinct color sets. The color sets representation size of FINIMAP could be improved using techniques described in the last version of Fulgor [10].

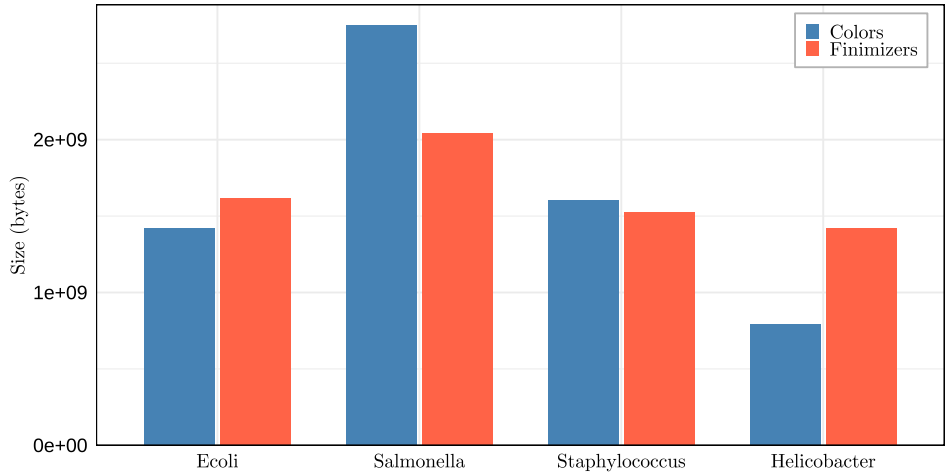



**Figure 10** Finimap index size divided in colors and finimizers representations.

**Table 3** Index size, in bits per $k$-mer, of all the tested tools with $k = 31$. blue: $< 2 \times best$, orange: $< 4 \times best$, magenta otherwise, where *best* is the smallest value in every column.

| | Ecoli | Salmonella | Staphylococcus | Helicobacter |
|---|---|---|---|---|
| Finimap | 71.20 | 60.32 | 230.24 | 95.68 |
| Raptor 19 | 129.76 | 239.44 | 384.88 | 25.28 |
| Kaminari 19 | 12.48 | 11.04 | 34.72 | 6.72 |
| Kaminari 21 | 14.24 | 12.64 | 40.00 | 8.00 |
| Kaminari 23 | 16.40 | 14.72 | 46.56 | 9.76 |
| Kaminari 25 | 19.20 | 17.36 | 55.04 | 12.16 |
| Kaminari 27 | 23.12 | 20.96 | 66.08 | 15.60 |
| Fulgor | 34.40 | 31.36 | 105.92 | 31.20 |
| mFulgor | 17.60 | 11.20 | 20.32 | 28.56 |
| dFulgor | 15.60 | 14.16 | 29.36 | 20.00 |
| mdFulgor | 12.56 | 9.20 | 16.48 | 22.16 |

**Table 4** Query time in seconds of threshold union queries with 50,000 positive queries of length 1,000, 1 thread, $k = 31$ and $\tau = 0.8$. Color code is blue: $< 2 \times best$, orange: $< 4 \times best$, magenta otherwise, where *best* is the smallest value in every column.

| | Ecoli | Salmonella | Staphylococcus | Helicobacter |
|---|---|---|---|---|
| Finimap | 40 | 103 | 101 | 30 |
| Raptor 19 | 23 | 73 | 67 | 11 |
| Kaminari 19 | 80 | 280 | 243 | 34 |
| Kaminari 21 | 86 | 313 | 271 | 39 |
| Kaminari 23 | 96 | 350 | 306 | 43 |
| Kaminari 25 | 110 | 403 | 351 | 50 |
| Kaminari 27 | 130 | 475 | 415 | 61 |
| Fulgor | 99 | 185 | 187 | 157 |
| mFulgor | 111 | 194 | 140 | 180 |
| dFulgor | 111 | 260 | 80 | 180 |
| mdFulgor | 86 | 187 | 127 | 138 |

# Selecting Chromosomes for Polygenic Traits: Algorithms and Complexity

**Or Zuk**
Department of Statistics and Data Science, The Hebrew University of Jerusalem, Israel

## Abstract

We define and study the problem of *genomic block selection* for multiple complex traits. In this problem, one constructs a genome by selecting different genomic parts (e.g. chromosomes) from different source genomes. The constructed genome is associated with a vector of polygenic scores, obtained by summing the polygenic scores of the different genomic parts, and the goal is to minimize a given loss function of this vector. The problem is motivated by several emerging technologies: chromosome substitution lines in crop breeding, where chromosomal segments from wild relatives are combined to improve polygenic traits such as yield and stress tolerance; chromosome transfer between yeast strains for optimizing complex industrial phenotypes; and chromosomal transplantation technologies in mammalian cells. We suggest and study several natural loss functions relevant for both quantitative and threshold traits, and show that the problem is NP-complete even for a single trait and two copies, yet only weakly so, being pseudo-polynomially solvable for any fixed number of traits. We propose three algorithms with complementary roles: a Branch-and-Bound algorithm that returns the certified global optimum for any monotone loss, a fast Block-Coordinate-Descent (BCD) heuristic with random restarts that applies to any loss, and a semidefinite-programming (SDP) relaxation that provides a certified lower bound on the optimal loss for quadratic losses, and hence an optimality-gap bound when paired with the BCD solution - empirically tight in our experiments. Using the infinitesimal model for genetic architecture, we further derive, for linear losses, a closed-form approximation for the expected gain of block selection relative to random selection across multiple traits. On yeast-scale simulations BCD matches the certified Branch-and-Bound optimum on 100% of threshold-loss instances at 466× the speed, attains a certified optimality gap of at most ≈10% of the SDP lower bound for stabilizing-loss instances, and the realized gain roughly matches the analytic prediction.

**2012 ACM Subject Classification** Mathematics of computing → Combinatorial optimization; Applied computing → Computational genomics; Applied computing → Computational genomics

**Keywords and phrases** polygenic scores, combinatorial optimization, genomic block selection, NP-hardness, semidefinite programming, synthetic genomics

**Digital Object Identifier** 10.4230/LIPIcs.WABI.2026.28

**Supplementary Material** *Software*: `https://github.com/orzuk/EmbryoSelectionCalculator`

**Funding** Supported by the Israeli Science Foundation (grants #2830/20 and #2392/22).

## 1 Introduction

In several breeding and biotechnology contexts, it is possible or becoming feasible to combine genomic parts from different source individuals into a single genome. In plant breeding, chromosome substitution lines are routinely created by introgressing chromosomal segments from wild relatives into elite cultivars, for example barley chromosome arms into wheat [16,33], in order to improve polygenic traits such as yield, disease resistance and stress tolerance. In yeast, the completion of the Sc2.0 synthetic genome project [34] and tools for chromosome transfer between strains [31, 38] open the possibility of constructing chimeric genomes optimized for complex industrial phenotypes such as ethanol tolerance and thermotolerance, which are controlled by dozens to hundreds of quantitative trait loci (QTL) [3, 4]. In


© Or Zuk;
licensed under Creative Commons License CC-BY 4.0

mammalian cells, high-fidelity chromosomal transplantation has been demonstrated [29, 30, 32], and programmable recombination systems now enable targeted, large-scale DNA rearrangements [12].

For highly polygenic traits where no individual gene has a large effect, selection of individual chromosomes or large-scale genomic regions from available genomes may complement local gene-editing approaches using the CRISPR-CAS9 system [21] that affect only one or a handful of genes. Chromosomal selection leverages existing genetic material without introducing new mutations, but comes with a trade-off: entire chromosomes carry both favorable and unfavorable alleles (linkage drag), whereas targeted editing can modify specific loci with precision.

Choosing which blocks to combine requires a quantitative predictor of each assembled genome's phenotype. Polygenic scores (PS) provide this, combining the contributions of multiple genomic alleles with coefficients typically fitted from Genome-Wide-Association-Studies (GWAS). GWAS originated for human complex traits [23, 25, 40] and has since been adopted in livestock [27], crops [11], and yeast [3, 31]; accuracy continues to improve with growing sample sizes and advances in statistical methods [9, 18, 28].

Here we formulate and study the *genomic block selection* problem for multiple quantitative traits. In this problem, multiple copies are available for each chromosome (or possibly a smaller genomic part), and based on these copies' PS we select one of them. These choices yield a genome assembled from the different chromosomal parts. We address the following two main questions:

1. How should the chromosomes be selected in order to maximize utility across multiple traits? When selecting for $T$ traits, each selection $\mathbf{c}$ of chromosomes leads to an overall genomic score vector $\boldsymbol{X}_{\mathbf{c}} \in \mathbb{R}^T$. A loss function $\mathcal{L} : \mathbb{R}^T \to \mathbb{R}$ is defined and our goal is to find the selection $\mathbf{c}$ minimizing the loss $\mathcal{L}(\boldsymbol{X}_{\mathbf{c}})$, and compare it to the loss obtained for random selection. When $C$ copies are available for each of the $M$ chromosomes, the total number of possible selections $C^M$ is exponential in $M$, hence the need to design efficient general algorithms for the problem.
2. What is the expected gain when using optimal chromosomal selection? How does it compare to the baseline, i.e. selecting genomes at random, as well as to the whole-genome selection procedure in which one of the $C$ source genomes is selected as a whole [22, 26]?

Our main contributions are threefold. First, we formulate the genomic block selection problem mathematically and establish its weak NP-hardness even in simple cases: a single-trait stabilizing loss and a two-trait all-pass loss, each with two copies. Second, we develop three algorithms with complementary theoretical and practical roles: (i) a Branch-and-Bound (B&B) algorithm that returns the certified global optimum for monotone losses, in which the number of Pareto-non-dominated partial selections retained at each stage empirically grows much slower than exhaustive search over the regimes we tested; (ii) a simple Block-Coordinate-Descent (BCD) heuristic with random restarts, applicable to any loss, utilizing the fact that updating a single block's selection touches only $T$ coordinates of the score vector and therefore runs in $O(M\,C\,T)$ per pass over all blocks; and (iii) a semidefinite-programming (SDP) relaxation that provides a certified lower bound on the loss for quadratic losses (empirically tight in our simulations), so it certifies the optimality gap of the BCD solution. Third, we estimate the expected gain achieved by chromosomal selection, deriving an analytic approximation for linear losses and validating the algorithms on a yeast-like simulation scenario.

# 2 The Chromosomal Selection Problem

## 2.1 Background and Selection Problems for Polygenic Scores

Consider a genome composed of $M$ distinct genomic blocks, typically whole chromosomes. For haploid organisms (e.g. haploid yeast with $M = 16$), each block corresponds to a single chromosome. Diploid organisms carry two homologs per chromosome, so a block may be a chromosome type ($M$) or an individual homolog ($2M$); a diploid yeast strain, for instance, has 16 chromosome types and 32 homologs. For polyploid organisms one may further distinguish sub-genomes and homologs. For example, hexaploid wheat has 7 chromosome groups across 3 homeologous sub-genomes, so $M$ can range from 7 (one block per chromosome group) through 21 ($7 \times 3$, one block per sub-genomic chromosome) to 42 ($7 \times 3 \times 2$, one block per homolog), depending on the resolution of selection. We associate with each block a polygenic-score vector representing the genetic contribution of the block to $T$ traits of interest. Suppose that we have $C$ distinct source genomes, hence $C$ copies are available for each block overall. Our goal is to select one copy for each block, possibly under constraints, yielding a full genome with desired properties in terms of the resulting polygenic score. Two natural selection schemes are: (i.) *whole-genome selection*, where one of the $C$ source genomes is chosen, so all selected blocks belong to the same genome; (ii.) *chromosomal (block) selection*, where different blocks can be drawn from different sources, yielding a chimeric genome and a search space of size $C^M$. Concrete instances include yeast strain panels (e.g. $C$ yeast strains, each with $M = 16$ chromosomes, give $C^{16}$ possible chimeras) and wheat breeding ($C$ donor lines, including wild relatives, provide chromosomal segments for each of the $M$ chromosome groups, and one seeks the best combination of segments to optimize multiple agronomic traits). Figure 1(a) illustrates whole-genome selection, in which a single source genome is chosen, while Figure 1(b) illustrates chromosomal selection from multiple source genomes.

## 2.2 Preliminaries

For a natural number $n \in \mathbb{N}$ denote the set $\{1, .., n\}$ by $[n]$. For two natural numbers $m \leq n$ denote the set $\{m, m+1, .., n\}$ by $[m, n]$. The vector of all zeros (ones) of length $n$ is denoted $\mathbf{0}_n$ ($\mathbf{1}_n$). For a polytope $\Delta \in \mathbb{R}^n$, define the projection operator $\mathbb{P}_\Delta(x) \equiv \arg\min_{y \in \Delta} \|y - x\|^2$.

**Tensors.** Let $\boldsymbol{X} \in \mathbb{R}_{m \times n \times p}$ be a $3^{rd}$-order tensor with elements $X_{ijk}$. We denote lower-dimensional fibers of a tensor using the $\bullet$ subscript. For example, $\boldsymbol{X}_{ij\bullet}$ denotes the vector $(X_{ij1}, .., X_{ijp}) \in \mathbb{R}^p$. Similarly, $\boldsymbol{X}_{\bullet j\bullet}$ is a matrix of size $m \times p$ containing all elements $X_{ijk}, \forall i \in [m], \forall k \in [p]$. We also describe sub-tensors obtained by taking subsets of the indices across each dimension. For example, $\boldsymbol{X}_{[i][j]\bullet}$ is a $3^{rd}$-order tensor of size $i \times j \times p$ obtained by taking the first $i$ and $j$ coordinates on the first and second dimension, respectively.

For a $3^{rd}$-order tensor $\boldsymbol{X} \in \mathbb{R}_{m \times n \times p}$ and a matrix $\mathbf{A} \in \mathbb{R}_{m \times n}$, define the *2-mode* tensor-by-matrix product [1, 24], as a matrix $[\boldsymbol{X} \times_2 \mathbf{A}] \in \mathbb{R}_{m \times p}$, with elements defined by:

$$[\boldsymbol{X} \times_2 \mathbf{A}]_{ik} = \sum_{j=1}^{n} X_{ijk} A_{ij}, \ \forall i \in [m], \forall k \in [p]. \tag{1}$$

**Gaussian distribution.** For the multivariate Gaussian distribution with mean $\mu$ and variance $\Sigma$, denote by $\phi(\mathbf{x}; \mu, \Sigma)$ and $\Phi(\mathbf{x}; \mu, \Sigma)$ the density function and cumulative distribution function, respectively. When $\mu, \Sigma$ are omitted, $\Phi$ and $\phi$ refer to the standard multivariate

Gaussian distribution with mean zero and identity covariance matrix. For a measurable set $\mathcal{A} \subset \mathbb{R}^T$ denote by $\Phi(\mathcal{A}; \mu, \Sigma)$ the probability of the set under the Gaussian distribution, i.e. $\Phi(\mathcal{A}; \mu, \Sigma) \equiv \int_{\mathcal{A}} \phi(\mathbf{x}; \mu, \Sigma) d\mathbf{x}$.

## 2.3 Chromosomal Selection for Multiple Traits

Let $\boldsymbol{X} \in \mathbb{R}_{M \times C \times T}$ be a $3^{rd}$-order tensor of polygenic scores, where $X_{ijk}$ denotes the score in chromosome $i$ of copy $j$ for trait $k$. Let $\mathbf{c} = (c_1, .., c_M) \in [C]^M$ be a selection vector. The associated selected polygenic vector is defined as $\boldsymbol{X}_{\mathbf{c}} \equiv \sum_{i=1}^{M} X_{ic_i\bullet} \in \mathbb{R}^T$, with $\boldsymbol{X}_{\mathbf{c}_k}$ denoting its $k$-th element $(\boldsymbol{X}_{\mathbf{c}})_k$, $\forall k \in [T]$. Our goal is to find the selected score vector $\boldsymbol{X}_{\mathbf{c}}$ minimizing a loss function of our choice. The *multi-trait chromosomal selection problem* is defined as follows:

▶ **Problem 1.** *Given a $3^{rd}$-order tensor of scores $\boldsymbol{X} \in \mathbb{R}_{M \times C \times T}$, and a loss function $\mathcal{L} : \mathbb{R}^T \to \mathbb{R}$, find a vector $\mathbf{c}^* \in [C]^M$ minimizing the loss: $\mathbf{c}^* \equiv \underset{\mathbf{c} \in [C]^M}{argmin} \mathcal{L}(\boldsymbol{X}_{\mathbf{c}})$.*

Table 1 lists several examples of natural loss functions of interest for both quantitative and threshold traits. The computational difficulty of Problem 1 above depends on the loss function $\mathcal{L}$. For example, when $\mathcal{L}$ is linear in the polygenic score vector $\boldsymbol{X}_{\mathbf{c}}$, we can select the optimal copy for each block independently, and the computational problem becomes trivial; this is the case when maximizing a weighted combination of quantitative traits. For non-linear losses (e.g. minimizing the overall failure probability across multiple threshold traits), selecting the best chromosome may be computationally challenging since the scores of all chromosomes must be considered jointly. In Section 3, we propose three algorithms for finding the optimal selection, applicable to general classes of loss functions.

Many natural loss functions are *monotone* in the scores vector, a property we exploit in some of our algorithms. Denote by $\mathbf{x} \preceq \mathbf{y}$ the coordinatewise product order on $\mathbb{R}^n$ (i.e. $x_i \leq y_i$ for all $i$), and write $\mathbf{x} \prec \mathbf{y}$ if $\mathbf{x} \preceq \mathbf{y}$ and $\mathbf{x} \neq \mathbf{y}$. A loss $\mathcal{L} : \mathbb{R}^n \to \mathbb{R}$ is *monotonically non-increasing* if $\mathbf{x} \preceq \mathbf{y}$ implies $\mathcal{L}(\mathbf{x}) \geq \mathcal{L}(\mathbf{y})$.

**Table 1** Example loss functions $\mathcal{L}(\boldsymbol{X}_{\mathbf{c}})$ and their properties. In all rows $\boldsymbol{X}_{\mathbf{c}} \in \mathbb{R}^T$ is the selected score vector and $\boldsymbol{X}_{\mathbf{c}_k}$ its $k$-th coordinate; $\mathbf{w} \in \mathbb{R}^T$ is a trait-weight vector. (i) The linear loss applies to quantitative traits (e.g. growth rate, stress tolerance, metabolite yield). (ii) The stabilizing loss [20, 35, 36] penalizes deviations of the resulting score from the trait optimum (set to zero w.l.o.g.); this is appropriate when a trait is already near its optimum and large deviations are harmful. (iii)–(iv) For $T$ threshold traits (e.g. disease susceptibility or failure to meet a quality standard), $D_k$ is a binary random variable (the status of trait $k$) whose distribution depends on the selected score $\boldsymbol{X}_{\mathbf{c}}$, and $\mathbf{D} = (D_1, \ldots, D_T)$; the crossing probability $P(D_k = 1 \mid \boldsymbol{X}_{\mathbf{c}})$ follows a liability-threshold model [14] and depends on the trait's prevalence, heritability, and the cross-trait residual covariance (full model in Appendix B). The threshold-linear loss minimizes a weighted sum of crossing probabilities; the all-pass loss takes the negative probability that no threshold is crossed.

| Loss | Formula | Monotone? | Convex? | Algorithms |
|---|---|---|---|---|
| Linear | $-\sum_{k=1}^{T} w_k \boldsymbol{X}_{\mathbf{c}_k}$, $\mathbf{w} \geq 0$ | YES | YES | Simple Selection |
| Stabilizing | $\sum_{k=1}^{T} w_k \boldsymbol{X}_{\mathbf{c}_k}^2$ | NO | YES | BCD; SDP (bound) |
| Threshold-linear | $\sum_{k=1}^{T} w_k P(D_k = 1 \mid \boldsymbol{X}_{\mathbf{c}})$ | YES | NO | B&B (exact) / BCD |
| All-pass | $-P(\mathbf{D} = \mathbf{0}_T \mid \boldsymbol{X}_{\mathbf{c}})$ | YES | NO | B&B (exact) / BCD |

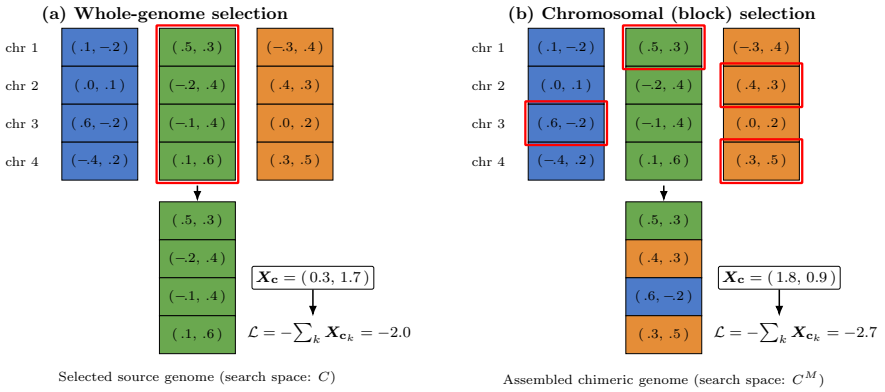



**Figure 1** Two paradigms for selecting a genome from $C$ source genomes, each providing one copy of $M$ chromosomes (or genomic blocks). The example shown uses $C = 3$ source genomes (blue, green, orange), $M = 4$ chromosomes, and $T = 2$ traits, with each cell labelled by its vector of polygenic scores $\boldsymbol{X}_{ij\bullet} \in \mathbb{R}^2$. The two paradigms differ in which subset is selected (red boxes). **(a)** Whole-genome selection picks the single source whose aggregated polygenic-score vector $\sum_i \boldsymbol{X}_{ij\bullet}$ is optimal; the red box wraps around the selected genome shown as an entire column (Source 2), so the output is a monochromatic genome with $\boldsymbol{X}_{\mathbf{c}} = (0.3, 1.7)$. **(b)** Chromosomal (block) selection assigns, jointly across chromosomes, a source $c_i \in [C]$ to each chromosome $i$, giving a chimeric genome $\boldsymbol{X}_{\mathbf{c}} = \sum_i \boldsymbol{X}_{ic_i\bullet}$ that minimizes the chosen loss $\mathcal{L}(\boldsymbol{X}_{\mathbf{c}})$; a red box wraps around each selected block. Here the loss is the linear loss $\mathcal{L}(\boldsymbol{X}_{\mathbf{c}}) = -\sum_k \boldsymbol{X}_{\mathbf{c}_k}$ with equal weights, i.e. maximizing the total score across both traits ($1.8 + 0.9 = 2.7$ versus $0.3 + 1.7 = 2.0$ for whole-genome selection). The output is a mosaic whose blocks are colored by their source of origin, with $\boldsymbol{X}_{\mathbf{c}} = (1.8, 0.9)$. Sources may represent, e.g. different yeast strains, crop donor lines, or engineered mammalian cells via chromosome transplantation.

### 2.3.1 The Gain due to Selection

▶ **Definition 2.** *For a tensor $\boldsymbol{X}$ of scores and a loss function $\mathcal{L}$, let $\mathbf{c} \sim \mathrm{Unif}([C]^M)$ be a uniformly random selection (each $c_i$ drawn i.i.d. uniformly from $[C]$), so that $\mathbb{E}_{\mathbf{c}}\mathcal{L}(\boldsymbol{X}_{\mathbf{c}}) = C^{-M}\sum_{\mathbf{c}\in[C]^M}\mathcal{L}(\boldsymbol{X}_{\mathbf{c}})$. We define the* Gain $G$ *due to chromosomal selection and the* Gain $G_e$ *due to whole-genome selection as the difference between this random-selection baseline and the optimal loss under each scheme:*

$$G(\boldsymbol{X};\mathcal{L}) \equiv \mathbb{E}_{\mathbf{c}}\mathcal{L}(\boldsymbol{X}_{\mathbf{c}}) - \mathcal{L}(\boldsymbol{X}_{\mathbf{c}^*})$$
$$G_e(\boldsymbol{X};\mathcal{L}) \equiv \mathbb{E}_{\mathbf{c}}\mathcal{L}(\boldsymbol{X}_{\mathbf{c}}) - \min_{j\in[C]} \mathcal{L}\Big(\sum_{i=1}^{M} \boldsymbol{X}_{ij\bullet}\Big). \tag{2}$$

The gain $G_e(\boldsymbol{X};\mathcal{L})$ is similar in spirit to previous definitions given in [22, 26] for human embryo selection, but generalized here for a general loss whereas [22, 26] defined the gain only for additive losses for quantitative and disease traits. By definition $G_e(\boldsymbol{X};\mathcal{L}) \le G(\boldsymbol{X};\mathcal{L})$; both are functions of the score tensor $\boldsymbol{X}$. Averaging over $\boldsymbol{X}$ drawn from a statistical model yields the *expected gains* $\mathbb{E}_{\boldsymbol{X}}G$ and $\mathbb{E}_{\boldsymbol{X}}G_e$. In Section 4 we derive closed-form approximations of the expected gain for linear loss functions.

## 3 Algorithms and Complexity

The tractability of Problem 1 depends on the loss. For the linear loss it decomposes across blocks, each optimized independently by picking $c_i = \arg\min_j -\mathbf{w}^t X_{ij\bullet}$ in overall $O(M\,C\,T)$ time. For general non-linear losses, we establish its computational hardness in Section 3.1. We

then develop three algorithms, Branch-and-Bound (Section 3.2), Block-Coordinate-Descent (Section 3.3), and a semidefinite relaxation (Section 3.4), together with a projected-gradient relaxation [7] baseline (Section 3.4.1) that our experiments found dominated by the other methods.

## 3.1 Computational Complexity

The optimization problem 1 has an exponential search space of size $C^M$. We characterize its complexity: the next two theorems show that it is NP-hard already in simple cases: the stabilizing loss at a single trait and the all-pass loss at two traits, each with two copies.

▶ **Theorem 3.** *The decision version of Problem 1 with the stabilizing-selection loss, namely deciding, given a tensor $\boldsymbol{X}$ and a threshold $\tau$, whether there exists $\mathbf{c} \in [C]^M$ with $\mathcal{L}(\boldsymbol{X}_{\mathbf{c}}) \le \tau$, is NP-complete, even for $T = 1$ and $C = 2$.*

**Proof.**

**NP-membership.** Given $\mathbf{c} \in [C]^M$, the score $\boldsymbol{X}_{\mathbf{c}} = \sum_{i=1}^{M} X_{ic_i\bullet}$ can be computed in $O(MT)$ time and the loss evaluated in $O(T)$ additional time, so the decision problem is in NP for any polynomial-time computable loss.

**NP-hardness.** We reduce from Partition[1]. Given positive integers $a_1, \ldots, a_M$ with $S = \sum_{i=1}^{M} a_i$, construct an instance with $C = 2$ copies, $T = 1$ trait, and the stabilizing loss $\mathcal{L}(x) = (x - S/2)^2$. Set the scores $X_{i11} = a_i$ and $X_{i21} = 0$ for all $i \in [M]$. For any selection vector $\mathbf{c} \in [2]^M$, the selected score is $\boldsymbol{X}_{\mathbf{c}} = \sum_{i:c_i=1} a_i$, and the loss is $\mathcal{L}(\boldsymbol{X}_{\mathbf{c}}) = (\sum_{i:c_i=1} a_i - S/2)^2$. This equals zero if and only if $\sum_{i:c_i=1} a_i = S/2$, i.e. the integers $\{a_i : c_i = 1\}$ form an exact partition, so a poly-time solver for our decision problem would decide Partition, hence the problem is NP-hard and therefore NP-complete. ◀

▶ **Theorem 4.** *Problem 1 with the all-pass loss $\mathcal{L}(\mathbf{x}) = -P(\mathbf{D} = \mathbf{0}_T | \boldsymbol{X}_{\mathbf{c}} = \mathbf{x})$ is NP-hard for $T \ge 2$ and $C = 2$. When the residual covariance is diagonal, the objective factorizes into univariate Gaussian CDFs and is efficiently evaluable.*

**Proof.** We reduce from Partition. Given positive integers $a_1, \ldots, a_M$ with $S = \sum a_i$, set $T = 2$, $C = 2$, with scores $X_{i11} = a_i$, $X_{i21} = 0$, $X_{i12} = 0$, $X_{i22} = a_i$, symmetric thresholds $z_{K_1} = z_{K_2} = S/2$, and residual covariance $\sigma^2 \mathbf{I}_2$ (i.e. conditionally independent trait residuals with common variance $\sigma^2$), so the joint conditional probability factorises across traits. Under the liability-threshold model (eq. (13), Appendix B), each factor is the per-trait pass probability $P(D_k = 0 \mid \boldsymbol{X}_{\mathbf{c}}) = \Phi\big((\boldsymbol{X}_{\mathbf{c}_k} - z_{K_k})/\sigma\big)$, the complement of the crossing probability. For a selection with $A = \{i : c_i = 1\}$ and $u = \sum_{i \in A} a_i$ we have $\boldsymbol{X}_{\mathbf{c}_1} = u$ and $\boldsymbol{X}_{\mathbf{c}_2} = S - u$, so the loss is $-\Phi\big(\frac{u - S/2}{\sigma}\big) \cdot \Phi\big(\frac{S/2 - u}{\sigma}\big)$. Since $\Phi(z)\Phi(-z) = \Phi(z)(1 - \Phi(z))$ is uniquely maximized at $z = 0$, the loss achieves its minimum $-1/4$ if and only if $u = S/2$, i.e. an exact partition exists. ◀

We next show that the hardness above is only *weak*, because the problem is pseudo-polynomially solvable for any fixed number of traits.

[1] Partition: given positive integers $a_1, \ldots, a_n$ with $S = \sum a_i$, decide whether there exists $A \subseteq [n]$ with $\sum_{i \in A} a_i = S/2$. Partition is weakly NP-complete [17].

▶ **Proposition 5** (Pseudo-polynomial algorithm; weak hardness)**.** *Fix the number of traits $T$, assume integer scores $X_{ijk}$, and let $R_k \equiv \sum_{i=1}^{M}(\max_j X_{ijk} - \min_j X_{ijk})$ be the range of achievable sums for trait $k$, with $R = \max_k R_k$. Then Problem 1 with any loss $\mathcal{L} : \mathbb{Z}^T \to \mathbb{R}$ evaluable in $O(T)$ time is solved exactly in $O(M\,C\,R^T)$ time and $O(R^T)$ space. Hence for every fixed $T$ the problem is pseudo-polynomial, so the NP-hardness of Theorems 3 and 4 is only* weak*, and bounded-precision score instances are solvable exactly in polynomial time.*

The proof, a dynamic program over the achievable score vectors, is given in Appendix A.2.

## 3.2 A Branch-and-Bound algorithm

In the Branch-and-Bound algorithm for selecting chromosomes *for monotone loss functions*, we grow a tree of all possible selected chromosomes, and at each level keep only leaves not dominated by other leaves. Finally, we evaluate the loss of all leaves at the last level. A tree of depth $b$ is represented as a collection of paths from the root to the leaves $\Gamma = \{\mathbf{c}^{(1)}, .., \mathbf{c}^{(m)}\}$ where each $\mathbf{c}^{(j)} \in [C]^b$ represents the choices of chromosomes in the first $b$ levels. The partial score sum is calculated: $\boldsymbol{X}_{\mathbf{c}^{(j)}} = \sum_{i=1}^{b} \boldsymbol{X}_{i\mathbf{c}_i^{(j)}\bullet}$, and dominated partial score vectors are pruned. Then, each of the remaining $\mathbf{c}^{(j)}$'s is expanded into $C$ paths of length $b+1$. A formal step-by-step description is shown in Algorithm 1. The computational burden of a single run is determined by the number of non-dominated partial-sum vectors retained at each stage $b$, out of $C^b$ possible partial selections. In the *worst case*, the Branch-and-Bound algorithm retains all of them, hence it may run in time exponential in $M$, as shown in the Appendix, Section A.3.

While the worst-case computational complexity of Algorithm 1 is exponential in $M$, the number of vectors considered may be far lower than $C^M$ in practice. Further pruning can be achieved by bounding the loss with the coordinatewise *envelopes* of the score tensor: let $\boldsymbol{X}_\vee$ ($\boldsymbol{X}_\wedge$) denote the vector obtained by summing, over all chromosomes $i$, the coordinatewise maximum (minimum) of $X_{ijk}$ across $j \in [C]$. Equivalently, writing $(\mathbf{A})_{\textcircled{\scriptsize$\vee$}}$ and $(\mathbf{A})_{\textcircled{\scriptsize$\wedge$}}$ for the column-wise maximum and minimum vectors of a matrix $\mathbf{A}$ (defined in Appendix A.1 and used in line 5 of Algorithm 1), we have $\boldsymbol{X}_\vee = \sum_i (\boldsymbol{X}_{i\bullet\bullet})_{\textcircled{\scriptsize$\vee$}}$ and $\boldsymbol{X}_\wedge = \sum_i (\boldsymbol{X}_{i\bullet\bullet})_{\textcircled{\scriptsize$\wedge$}}$. Since $\boldsymbol{X}_\wedge \preceq \boldsymbol{X}_{\mathbf{c}} \preceq \boldsymbol{X}_\vee$ for every feasible selection $\mathbf{c}$, and $\mathcal{L}$ is monotonically non-increasing, for the optimal selected score $\boldsymbol{X}_\star \equiv \boldsymbol{X}_{\mathbf{c}^\star}$ we obtain

$$\mathcal{L}(\boldsymbol{X}_\vee) \;\le\; \mathcal{L}(\boldsymbol{X}_\star) \;\le\; \mathcal{L}(\boldsymbol{X}_\wedge). \tag{3}$$

We call this an *envelope bound*: any partial-sum vector whose optimistic completion (the partial sum plus the remaining-block envelope $\sum_{l>i}(\boldsymbol{X}_{l\bullet\bullet})_{\textcircled{\scriptsize$\vee$}}$) yields a loss exceeding the feasible upper bound $\widehat{L}$ (the loss of the whole-genome optimum) cannot be optimal and is discarded. We refer to this discarding step as *envelope-pruning*.

For simulated problem instances, using the per-chromosome score model in eq. (11), the *average* number of leaves at each stage $b$ was far below the $C^b$ worst case over the regimes we tested (roughly $C^{b/2}$ for $T = 5$; see Figure 2(a,b)), which makes Algorithm 1 practical for problem instances of yeast and crop scale (e.g. $C = 2, M \le 21$).

### 3.2.1 Divide-and-conquer

We can improve the speed of our algorithm, by dividing the $M$ chromosomes into groups, optimizing each of them separately, and then combining the solution in a manner where sub-vectors that cannot be extended to the optimal solution are filtered out. This procedure significantly improves performance while still being guaranteed to yield an optimal solution for monotone losses. Due to its technical details, it is described in Appendix A.4.

**Algorithm 1** Choosing Best Genome (Branch-and-Bound).

**Input:** $\boldsymbol{X}$, a $3^{rd}$-order tensor of polygenic scores.
**Parameters:** $\mathcal{L}$, a monotonically non-increasing loss function.

Γ ← {∅} with partial sum $\mathbf{0}_T$; $\widehat{L} \leftarrow \min_{j\in[C]} \mathcal{L}(\sum_i \boldsymbol{X}_{ij\bullet})$ (whole-genome optimum).
**for** $i = 1$ to $M$ **do**
  EXPAND: $\Gamma \leftarrow \{(\mathbf{c}, j) : \mathbf{c} \in \Gamma,\ j \in [C]\}$ with partial sums $\boldsymbol{X}_{\mathbf{c}'} = \boldsymbol{X}_{\mathbf{c}} + X_{ij\bullet}$.
  PARETO-PRUNE: retain $\mathbf{c}' \in \Gamma$ only if $\nexists\, \tilde{\mathbf{c}} \in \Gamma$ with $\boldsymbol{X}_{\mathbf{c}'} \prec \boldsymbol{X}_{\tilde{\mathbf{c}}}$.
  (OPTIONAL) ENVELOPE-PRUNE: discard $\mathbf{c}' \in \Gamma$ if $\mathcal{L}\big(\boldsymbol{X}_{\mathbf{c}'} + \sum_{l>i} (\boldsymbol{X}_{l\bullet\bullet})_{\textcircled{\vee}}\big) > \widehat{L}$.
Output $\mathbf{c}^\star = \arg\min_{\mathbf{c}\in\Gamma} \mathcal{L}(\boldsymbol{X}_{\mathbf{c}})$.

## 3.3 Block Coordinate Descent

Algorithm 1 (Branch-and-Bound) is inapplicable for non-monotone loss functions, and even for monotone losses its worst-case complexity is exponential in $M$. We therefore describe a simple *Block-Coordinate-Descent* (BCD) heuristic that applies to any loss and whose per-sweep complexity is linear in $M, C, T$.

The key observation is that updating a single block's selection $c_i$ while holding all other $c_{i'}$ fixed changes only the contribution of block $i$ to the aggregate score vector $\boldsymbol{X}_{\mathbf{c}} \in \mathbb{R}^T$. Concretely, write $\boldsymbol{X}_{\mathbf{c}} = \boldsymbol{X}_{\mathbf{c}}^{(-i)} + \boldsymbol{X}_{ic_i\bullet}$ where $\boldsymbol{X}_{\mathbf{c}}^{(-i)} \equiv \sum_{i'\neq i} \boldsymbol{X}_{i'c_{i'}\bullet}$ is the partial sum over all blocks except $i$. For each $j \in [C]$, evaluating $\mathcal{L}(\boldsymbol{X}_{\mathbf{c}}^{(-i)} + \boldsymbol{X}_{ij\bullet})$ costs only $O(T)$ once $\boldsymbol{X}_{\mathbf{c}}^{(-i)}$ is known. A full sweep over all blocks therefore costs $O(M\,C\,T)$ per iteration, and BCD typically converges to a 1-swap local optimum in $O(1)$ sweeps. In practice we repeat the procedure from $R$ random restarts and keep the best result; see Algorithm 2.

**Algorithm 2** Choosing Best Genome (Block Coordinate Descent).

**Input:** $\boldsymbol{X}$, a $3^{rd}$-order tensor of polygenic scores.
**Parameters:** $\mathcal{L}$, any loss function; $R$, the number of restarts.

Build $R$ initializations: i. whole-genome best $(j', \ldots, j')$ with $j' = \arg\min_j \mathcal{L}(\sum_i \boldsymbol{X}_{ij\bullet})$; ii. per-block greedy $(\arg\min_j \mathcal{L}(\boldsymbol{X}_{1j\bullet}), \ldots, \arg\min_j \mathcal{L}(\boldsymbol{X}_{Mj\bullet}))$; iii. $R-2$ uniformly random selections. Set $\mathbf{c}_{\text{best}} \leftarrow \emptyset$, $L_{\text{best}} \leftarrow +\infty$.
**for** each initialization $\mathbf{c}^{(0)}$ **do**
  $\mathbf{c} \leftarrow \mathbf{c}^{(0)}$; $\boldsymbol{X}_{\mathbf{c}} \leftarrow \sum_i \boldsymbol{X}_{ic_i\bullet}$; *changed* ← TRUE.
  **while** *changed* **do**
    *changed* ← FALSE.
    **for** $i = 1, \ldots, M$ **do**
      $\boldsymbol{X}_{\mathbf{c}}^{(-i)} \leftarrow \boldsymbol{X}_{\mathbf{c}} - \boldsymbol{X}_{ic_i\bullet}$; $j^\star \leftarrow \arg\min_{j\in[C]} \mathcal{L}(\boldsymbol{X}_{\mathbf{c}}^{(-i)} + \boldsymbol{X}_{ij\bullet})$.
      **if** $\mathcal{L}(\boldsymbol{X}_{\mathbf{c}}^{(-i)} + \boldsymbol{X}_{ij^\star\bullet}) < \mathcal{L}(\boldsymbol{X}_{\mathbf{c}})$ **then**
        $c_i \leftarrow j^\star$; $\boldsymbol{X}_{\mathbf{c}} \leftarrow \boldsymbol{X}_{\mathbf{c}}^{(-i)} + \boldsymbol{X}_{ij^\star\bullet}$; *changed* ← TRUE.
  **if** $\mathcal{L}(\boldsymbol{X}_{\mathbf{c}}) < L_{\text{best}}$ **then**
    $\mathbf{c}_{\text{best}} \leftarrow \mathbf{c}$; $L_{\text{best}} \leftarrow \mathcal{L}(\boldsymbol{X}_{\mathbf{c}})$.
Output $\mathbf{c}_{\text{best}}$.

BCD has no global-optimality guarantee: it converges to a local optimum with respect to single-block swaps. However, the structural decoupling (independent blocks given the running score vector) makes it efficient enough that random restarts explore the landscape effectively in our yeast-scale simulations, where $R = 10$ restarts sufficed (Section 5).

## 3.4 A Semidefinite Relaxation

For the (non-monotone) stabilizing selection loss $\mathcal{L}(\boldsymbol{X}_{\mathbf{c}}) = \sum_{k=1}^{T} w_k \boldsymbol{X}_{\mathbf{c}_k}^2$, the Branch-and-Bound algorithm of Section 3.2 does not apply and BCD has no optimality guarantee. We first review a simpler continuous relaxation that serves as a baseline, then develop a Shor-type semidefinite relaxation that provides a *certified lower bound* on the optimal loss, hence an *upper bound on the achievable gain*. Combined with a feasible heuristic such as BCD, the latter yields a certified optimality gap.

### 3.4.1 A simpler baseline: projected-gradient relaxation

As a standard baseline we also consider a continuous-relaxation approach: encode each selection $c_i \in [C]$ by a one-hot vector $\mathbf{C}_{i\bullet}$ and relax $\mathbf{C}_{i\bullet} \in \Delta_C$ (the $C$-simplex); the aggregate score becomes $\boldsymbol{X}_{\mathbf{c}} = [\boldsymbol{X} \times_2 \mathbf{C}]^t \mathbf{1}_M$. We minimize $\mathcal{L}\big([\boldsymbol{X} \times_2 \mathbf{C}]^t \mathbf{1}_M\big)$ over $\mathbf{C} \in \Delta_C^M$ by projected gradient descent with per-row simplex projection [7] and recover an integer solution by row-wise argmax; convergence guarantees apply only when $\mathcal{L}$ is convex [10]. For the stabilizing loss, a closed-form solution of the relaxation's Karush–Kuhn–Tucker (KKT) optimality conditions gives a Lagrangian variant. Empirically the rounded loss was worse than the trivial baseline of picking copy 1 for every block (Section 5): the relaxed optimum sits in the simplex interior, and argmax rounding then picks essentially-arbitrary copies. This failure mode directly motivates the semidefinite relaxation below, which enforces an integer-feasible structure and yields a certified lower bound on the integer loss.

### 3.4.2 The semidefinite relaxation

Let $\mathbf{q} = \mathrm{vec}(\mathbf{C}) \in \{0,1\}^{MC}$ be the vectorized one-hot selection matrix ($C_{ij} = 1$ iff $c_i = j$; vec as defined in Appendix A.1), so $q_{(i-1)C+j} = C_{ij}$. The stabilizing loss is then the quadratic form $\mathcal{L}(\mathbf{q}) = \mathbf{q}^t \mathbf{A} \mathbf{q}$, where $\mathbf{A} = \sum_{k=1}^{T} w_k \mathbf{x}_k \mathbf{x}_k^t$ and $\mathbf{x}_k = \mathrm{vec}(\boldsymbol{X}_{\bullet\bullet k})$. Following Shor [39], we lift the binary problem with the bordered matrix $\boldsymbol{W} = \begin{pmatrix} 1 & \mathbf{q}^t \\ \mathbf{q} & \mathbf{Q} \end{pmatrix} \in \mathbb{R}_{(MC+1)\times(MC+1)}$. A genuine selection corresponds to a *rank*-1 $\boldsymbol{W} \succeq 0$ with $\mathbf{Q} = \mathbf{q}\mathbf{q}^t$, so that $\mathcal{L}(\mathbf{q}) = \mathrm{tr}(\mathbf{A}\mathbf{Q})$; imposing this rank-1 condition, the binary constraint $\mathrm{diag}(\mathbf{Q}) = \mathbf{q}$ (valid since $q_i^2 = q_i$ for binary $q_i$), and the per-block row-sum constraints reformulates Problem 1 exactly. Dropping the (non-convex) rank-1 constraint yields the SDP relaxation

$$\begin{aligned} \min_{\boldsymbol{W}} \quad & \mathrm{tr}(\mathbf{A}\mathbf{Q}) \\ \text{subject to} \quad & \boldsymbol{W} \succeq 0, \quad W_{1,1} = 1, \quad \mathrm{diag}(\mathbf{Q}) = \mathbf{q}, \\ & \sum_{j=1}^{C} q_{(i-1)C+j} = 1 \text{ for each } i \in [M]. \end{aligned} \tag{4}$$

A feasible $\boldsymbol{W}$ need not be rank 1 (so the optimum may be attained at a $\boldsymbol{W}$ that is not a genuine selection); but every rank-1 integer-feasible $\boldsymbol{W}$ remains feasible, so the optimal value $L^\star_{\mathrm{SDP}}$ is a *lower bound* on the integer minimum loss, $L^\star_{\mathrm{SDP}} \le \min_{\mathbf{c}} \mathcal{L}(\boldsymbol{X}_{\mathbf{c}})$. The relaxation is solved in polynomial time using interior-point methods (we use CSDP [5]). Given any feasible integer solution $\hat{\mathbf{c}}$ (e.g. from BCD), the certified optimality gap is $\mathcal{L}(\boldsymbol{X}_{\hat{\mathbf{c}}}) - L^\star_{\mathrm{SDP}}$. The procedure is summarised in Algorithm 3.

The SDP is used solely to certify the lower bound; a feasible integer selection could be recovered by block-wise argmax rounding of $\mathbf{q}^\star$, but empirically BCD with random restarts (Algorithm 2) produced a stronger feasible solution at much lower computational cost in our experiments (Section 5), so we pair the SDP bound with the BCD solution.

**Algorithm 3** Certified Lower Bound via Semidefinite Relaxation.

**Input:** $\boldsymbol{X}$, a $3^{rd}$-order tensor of polygenic scores; weights $\mathbf{w}$.
**Output:** $L^{\star}_{\mathrm{SDP}}$, a lower bound on the optimal stabilizing loss.

1: $\mathbf{A} \leftarrow \sum_{k=1}^{T} w_k \,\mathrm{vec}(\boldsymbol{X}_{\bullet\bullet k})\,\mathrm{vec}(\boldsymbol{X}_{\bullet\bullet k})^t$.
2: Solve the SDP in eq. (4) for an optimal $\boldsymbol{W}^{\star} = \begin{pmatrix} 1 & (\mathbf{q}^{\star})^t \\ \mathbf{q}^{\star} & \mathbf{Q}^{\star} \end{pmatrix}$, using an interior-point solver such as CSDP.
3: Output the bound $L^{\star}_{\mathrm{SDP}} = \mathrm{tr}(\mathbf{A}\mathbf{Q}^{\star})$.

## 4 Expected Gain: Analytic Prediction

To anchor the simulations that follow, we first derive a closed-form prediction for the expected gain under a population-level statistical model. Suppose the score tensor $\boldsymbol{X}$ is drawn from the matrix-Normal model of Appendix B, in which the score vector for each chromosome $i$ has covariance $\alpha_i^2\,\boldsymbol{\Sigma}^{(\boldsymbol{X})}$ with $\sum_i \alpha_i^2 = 1$, and blocks are independent. For the linear loss $\mathcal{L} = -\mathbf{w}^t \boldsymbol{X}_{\mathbf{c}}$ with $\mathbf{w} \succeq \mathbf{0}_T$ and i.i.d. source genomes ($\boldsymbol{\Sigma}^{(\boldsymbol{C})} = \mathbf{I}_C$), standard extreme-value arguments [22] give whole-genome and chromosomal expected gains

$$\begin{aligned} \mathbb{E}_{\boldsymbol{X}} G_e &\approx 0.77\sqrt{\mathbf{w}^t \boldsymbol{\Sigma}^{(\boldsymbol{X})} \mathbf{w}}\sqrt{\log C}, \\ \mathbb{E}_{\boldsymbol{X}} G &\approx \Big(\sum\nolimits_{i=1}^{M} \alpha_i\Big)\, 0.77\sqrt{\mathbf{w}^t \boldsymbol{\Sigma}^{(\boldsymbol{X})} \mathbf{w}}\sqrt{\log C}, \end{aligned} \tag{5}$$

so the ratio of expected gains is $\sum_i \alpha_i$. Using chromosome lengths as a proxy for $\alpha_i^2$ (Table 2, Appendix B), this gives a predicted $\sum_i \alpha_i \approx 3.89$-fold improvement for yeast ($M = 16$) and $\approx 4.68$-fold for human ($M = 23$, 22 autosomes plus the X chromosome). For non-linear losses the closed form no longer applies and we evaluate the gain numerically below.

## 5 Simulation Results

To examine the utility of the algorithms, we have implemented them in an R package available at *github* (see Appendix C). We simulated chromosomal scores from the statistical model of Appendix B, a Matrix-Gaussian distribution [19], with parameters inspired by a yeast-like scenario: $M = 16$ chromosomes (relative lengths in Table 2), varying $C$, and $T = 5$ quantitative traits under a threshold model. Each trait has a "failure" probability of 0.1 (analogous to a disease prevalence or a probability of falling below a quality threshold), a narrow-sense heritability $h^2 = 50\%$, and polygenic scores explaining 20% of the liability variance; the residual (non-score) covariance $\boldsymbol{\Sigma}^{(\boldsymbol{Y})} + \boldsymbol{\Sigma}^{(\boldsymbol{e})}$ therefore has diagonal 0.8, with off-diagonal entries set to 0.65 (moderate cross-trait residual correlation).

We used the sum of threshold-trait probability loss function with equal weights, and the (minus) probability of passing all thresholds simultaneously. The baseline loss under random selection was, as expected, $0.1 \times 5 = 0.5$ for the first loss. For the all-pass criterion, the baseline pass probability $P(\mathbf{D} = \mathbf{0}_T)$ was 0.72 (so the corresponding loss is $-0.72$), above $0.9^5 \approx 0.59$ that would hold under trait independence.

**Tree-pruning behavior of Branch-and-Bound (Figure 2a,b).** For a single simulated instance with $M = 16$, $C = 2$, $T = 5$, we tracked the number of Pareto-optimal partial-sum vectors retained at each stage of Algorithm 1 (Branch-and-Bound) and Algorithm 4 (divide-and-conquer). Branch-and-Bound retained $\approx$ 3,000 vectors at the last stage, and the divide-and-conquer variant retained only $\approx 100$, roughly two orders of magnitude fewer than the exhaustive $2^{16} = 65{,}536$ leaves. Averaging over 100 simulations for varying $M \in [2, 16]$,

the final Pareto-set size for divide-and-conquer remained in the few-hundreds range even at $M = 16$. The favorable average-case behavior can be understood through the theory of random Pareto fronts: when $C^b$ i.i.d. vectors in $\mathbb{R}^T$ are drawn, the expected number of Pareto-optimal points is $O\big((\log C^b)^{T-1}\big) = O\big(b^{T-1}(\log C)^{T-1}\big)$ [2], which for fixed $T$ grows polynomially in $b$. The partial-sum vectors in Algorithm 1 are not i.i.d., but their approximate independence under the per-chromosome Matrix-Normal distribution in eq. (11) suggests a similar scaling, consistent with the empirically observed growth of $\approx C^{b/2}$ non-dominated vectors for $T = 5$. For $T \gg 5$ the Pareto front grows and the tree may become prohibitively large; in that regime BCD or the SDP-bound become attractive.

**Threshold loss (Figure 2c).** For the weighted-sum threshold loss with $M = 16$, $T = 5$ and $C \in [2, 10]$, we ran 12 independent simulations and compared the certified Branch-and-Bound optimum[2] (Algorithm 4) against (i.) Block-Coordinate-Descent with $R = 10$ restarts (Algorithm 2) and (ii.) whole-genome selection ($\min_{j\in[C]} \mathcal{L}(\sum_i \boldsymbol{X}_{ij\bullet})$). BCD matched the Branch-and-Bound optimum on all 108 simulated instances (12 replicates per $C$, $C \in [2, 10]$); mean BCD running time was 0.01 s vs. 4.8 s for Branch-and-Bound at $C = 10$ (all timings in this section use the divide-and-conquer variant, Algorithm 4); summed over all $C \in [2, 10]$, BCD's total running time was 466× shorter (the per-$C$ speed-up grows with $C$ as Branch-and-Bound slows). To confirm these conclusions are not specific to $M = 16, T = 5$, we repeated this comparison across $M \in \{8, 16, 23\}$ and $T \in \{2, 3, 5\}$ ($C = 5$, 10 replicates each; 90 instances); BCD with $R = 10$ restarts again matched the certified Branch-and-Bound optimum on every instance. Branch-and-Bound time grew with problem size, reaching ≈1.7 s at $M = 23$ and over an hour per instance at $T = 6$ (as the Pareto front expands), while BCD stayed ≈0.01 s throughout. Chromosomal selection achieved roughly 3× the expected gain of whole-genome selection at every value of $C$ tested. This empirical ratio is consistent with the analytic prediction of $\sum_i \alpha_i \approx 3.89$ for yeast derived in eq. (5) for the linear loss: although the threshold loss is non-linear, over the simulated range it behaves approximately linearly in the relevant score directions, giving a similar block-additive geometry, so the ratio of expected gains tracks the linear prediction closely. The full agreement between BCD and B&B suggests that instances drawn from the matrix-Normal model in eq. (11) induce a benign optimization landscape for BCD, where $R = 10$ restarts suffice with high probability. On adversarial inputs (e.g. PARTITION-style constructions, cf. Theorem 3) BCD's 1-swap neighbourhood is no longer sufficient and the worst-case gap can be large. The projected-gradient baseline (Section 3.4.1) was again worse than the trivial copy-1 selection and is not displayed in Figure 2c.

**Stabilizing loss (Figure 2d).** For the (non-monotone) stabilizing loss with the same yeast parameters and 30 simulations per $C$, we computed (i) the BCD solution (Algorithm 2) and (ii) the SDP-guaranteed lower bound (Algorithm 3). Across all 270 simulated instances (30 replicates per $C$, $C \in [2, 10]$) the SDP bound was below BCD's feasible loss. Averaging across replicates, BCD's certified optimality gap, the relative excess of its feasible loss over the SDP lower bound $(\mathcal{L}_{\mathrm{BCD}} - L^\star_{\mathrm{SDP}})/L^\star_{\mathrm{SDP}}$, was 2.5%, 4.1%, 10.1% and 8.6% at $C = 2, 3, 4, 5$, respectively; at $C \geq 6$ the SDP bound and the BCD loss both approach zero and the absolute gap shrinks below $10^{-3}$. Note that Figure 2(d) plots the resulting expected *gain*, on whose scale this gap is under 1%, so the BCD and SDP-ceiling curves are visually indistinguishable.

[2] Our Branch-and-Bound implementation supports an optional heuristic cap of $B$ retained Pareto-optimal vectors per stage (Appendix C); for all experiments reported in this section the cap was set to $B = 10^7$ and was never reached, hence the reported solutions are genuine global optima of Problem 1.

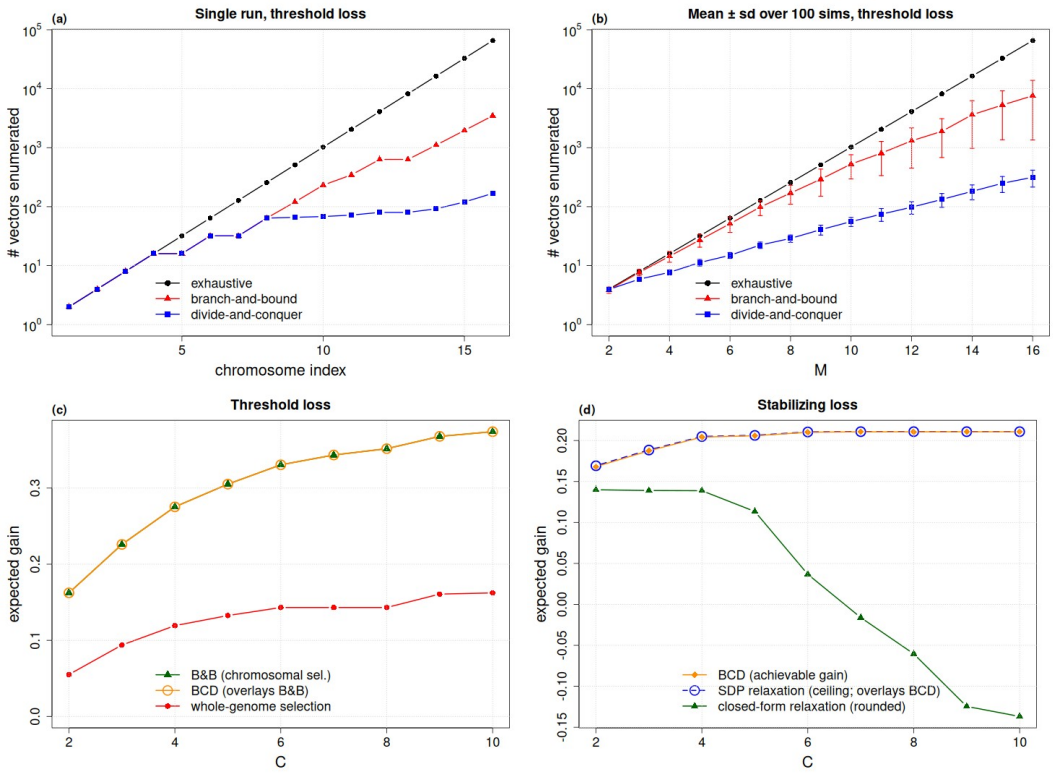



**Figure 2** (a.) Number of Pareto-optimal partial-sum vectors retained at each stage (y-axis) vs. the chromosome index (x-axis), for a single run with $M = 16$ chromosomes (yeast), $C = 2$ source strains, and $T = 5$ traits, using the threshold loss. Black: exhaustive search ($C^b$); red: Algorithm 1 (Branch-and-Bound); blue: the divide-and-conquer variant (Algorithm 4), here using $b = 2$ sub-blocks of $\lfloor M/2 \rfloor$ chromosomes each (two halves of 8 at $M = 16$). Divide-and-conquer retains $\approx 100$ vectors at $M = 16$, roughly two orders of magnitude below exhaustive. (b.) Mean $\pm$ one standard deviation of the final Pareto-set size across 100 simulations, for varying $M \in [2, 16]$. Both Branch-and-Bound (red) and divide-and-conquer (blue) grow much more slowly than exhaustive search over the tested regime; divide-and-conquer remains in the few-hundreds range at $M = 16$. (c.) Expected gain on the threshold loss vs. the number of source strains $C$, for yeast parameters ($M = 16$, $T = 5$, 12 simulations per $C$). Green: Branch-and-Bound (certified global optimum); orange: BCD (Algorithm 2, $R = 10$ restarts), which matched the Branch-and-Bound optimum on every simulated instance and is visually indistinguishable; red: whole-genome selection. Chromosomal selection (B&B or BCD) achieves $\approx 3\times$ the expected gain of whole-genome selection. (d.) Expected gain on the (non-monotone) stabilizing loss vs. $C$, for the same parameters and 30 simulations per $C$. Orange: BCD (achievable gain); blue dashed (x markers): SDP relaxation (Algorithm 3) value, an upper bound on the achievable gain; green: closed-form Lagrangian relaxation followed by argmax rounding. BCD's certified optimality gap relative to the SDP loss bound is 2–10% for $C \leq 5$; on the gain scale plotted here that gap is under 1%, so the BCD (orange) and SDP (blue) curves nearly coincide. The closed-form relaxation degrades sharply and yields negative gain at $C \geq 6$.

In contrast, the closed-form Lagrangian relaxation (dropping the nonneg constraint, solving the row-sum-constrained quadratic in closed form, then rounding by row-wise argmax; plotted in green) degrades sharply with $C$ and yields negative gain (worse than the trivial baseline) at $C \geq 6$, illustrating the failure mode of simplex-relaxation+rounding for this loss.

**Practical recommendation.** Based on these results, we recommend BCD with random restarts as the default algorithm in practice. Branch-and-Bound (Section 3.2) should be used when a certified global optimum is required for a monotone loss, and is tractable at the yeast/crop scale ($M \lesssim 21$, $C \lesssim 10$); at larger scale (e.g. when blocks are taken to be finer-grained chromosomal segments, $M \gg 21$, or when the Pareto front blows up at large $T$), the certified branch is infeasible and BCD becomes the only practical option. The SDP lower bound (Section 3.4) provides an optimality-gap certificate for quadratic losses; its complexity grows as $(MC)^{O(1)}$ via interior-point methods, so it remains usable beyond the regime where B&B applies, though it too eventually scales out and one must fall back on BCD alone.

## 6 Discussion

We have defined and formulated the genomic block selection problem and provided three algorithms with complementary roles. Our Branch-and-Bound algorithm (Algorithm 1) returns the certified global optimum for any monotone loss; despite worst-case exponential complexity, the divide-and-conquer variant (Algorithm 4) kept the retained Pareto set in the few-hundreds range at $M = 16$, while the plain version retained $\approx$ 3,000 vectors, both well within reach for yeast and crop scenarios ($M \leq 21$). The Block-Coordinate-Descent heuristic (Algorithm 2) applies to any loss, runs in milliseconds on yeast-scale instances, and was empirically optimal on every threshold-loss instance we tested (all 108 simulated instances). For the non-monotone stabilizing loss the SDP relaxation (Algorithm 3) provides a certified lower bound that, combined with the BCD solution, yielded a certified optimality gap of $\lesssim 10\%$ for $C \leq 5$ in our simulations. The standard projected-gradient relaxation (Section 3.4.1), included as a baseline, was strictly dominated by BCD on both monotone and non-monotone losses in our setup. Establishing problem-instance-dependent approximation guarantees for BCD and tightening the SDP at larger $C$ are interesting directions for future research.

The problem is motivated by several current and emerging technologies. In crop breeding, chromosome substitution lines from wild relatives are routinely used to improve polygenic traits in wheat and other polyploid crops [16, 33], yet the optimal combination of donor segments for multiple traits is typically chosen by breeder intuition rather than formal optimization; our algorithms apply directly once polygenic scores are available. In yeast, the Sc2.0 synthetic genome project [34] and panels of sequenced strains [31] make chimeric genomes for industrial phenotypes feasible (across dozens of yeast growth traits, including high-temperature growth, mapped QTL explain a median of $\approx 73\%$ of the additive heritability [3]). In mammalian cells, high-fidelity chromosome transplantation has been demonstrated [32], suggesting future relevance to animal models and potentially human applications, subject to technological and ethical constraints.

Our framework assumes additive polygenic scores across chromosomes, a consequence of the infinitesimal model; cross-chromosome epistasis, particularly between divergent source genomes, could reduce the realized gain, so eq. (5) should be read as an optimistic benchmark under the additive model rather than a worst-case bound. Our empirical evaluation, moreover, uses scores simulated from the matrix-Normal model (calibrated to real yeast chromosome

lengths and heritabilities); validating the algorithms on empirical per-chromosome polygenic scores (e.g. from sequenced yeast strain panels or crop populations) is an important next step. Extending the framework to higher-order chromosomal interactions, finer-grained selection units (chromosomal segments rather than whole chromosomes), and nonlinear polygenic scores are all important directions for future work.

## References


1 B.W. Bader and T.G. Kolda. Algorithm 862: Matlab tensor classes for fast algorithm prototyping. *ACM Transactions on Mathematical Software (TOMS)*, 32(4):635–653, 2006. doi:10.1145/1186785.1186794.

2 O. Barndorff-Nielsen and M. Sobel. On the distribution of the number of admissible points in a vector random sample. *Theory of Probability & Its Applications*, 11(2):249–269, 1966. doi:10.1137/1111020.

3 J.S. Bloom, J. Boocock, S. Treusch, M.J. Sadhu, L. Day, H. Oates-Barker, and L. Kruglyak. Rare variants contribute disproportionately to quantitative trait variation in yeast. *eLife*, 8:e49212, 2019. doi:10.7554/eLife.49212.

4 J.S. Bloom, I.M. Ehrenreich, W.T. Loo, T.-L.V. Lite, and L. Kruglyak. Finding the sources of missing heritability in a yeast cross. *Nature*, 494(7436):234–237, 2013.

5 B. Borchers. CSDP, a C library for semidefinite programming. *Optimization Methods and Software*, 11(1–4):613–623, 1999.

6 B. Bulik-Sullivan, H.K. Finucane, V. Anttila, A. Gusev, F.R. Day, P-R. Loh, L. Duncan, J.RB. Perry, N. Patterson, E.B. Robinson, et al. An atlas of genetic correlations across human diseases and traits. *Nature genetics*, 47(11):1236–1241, 2015.

7 Y. Chen and X. Ye. Projection onto a simplex. *arXiv preprint*, 2011. arXiv:1101.6081.

8 J.M. Cherry, E.L. Hong, C. Amundsen, R. Balakrishnan, G. Binkley, E.T. Chan, et al. Saccharomyces genome database: the genomics resource of budding yeast. *Nucleic Acids Research*, 40(D1):D700–D705, 2012.

9 S.W. Choi, T. S-H. Mak, and P.F. O'Reilly. Tutorial: a guide to performing polygenic risk score analyses. *Nature Protocols*, 15(9):2759–2772, 2020.

10 R. Correa and C. Lemaréchal. Convergence of some algorithms for convex minimization. *Mathematical Programming*, 62(1):261–275, 1993. doi:10.1007/BF01585170.

11 J. Crossa, P. Pérez-Rodríguez, J. Cuevas, O. Montesinos-López, D. Jarquín, G. de los Campos, J. Burgueño, J.M. González-Camacho, S. Pérez-Elizalde, Y. Beyene, et al. Genomic selection in plant breeding: methods, models, and perspectives. *Trends in Plant Science*, 22(11):961–975, 2017.

12 M.G. Durrant, N.T. Perry, J.J. Pai, A.R. Jangid, J.S. Athukoralage, M. Hiraizumi, J.P. McSpedon, A. Pawluk, H. Nishimasu, S. Konermann, and P.D. Hsu. Bridge RNAs direct programmable recombination of target and donor DNA. *Nature*, 630:984–993, 2024. doi:10.1038/s41586-024-07552-4.

13 D. Eddelbuettel and R. François. Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, 40(8):1–18, 2011. doi:10.18637/jss.v040.i08.

14 D.S. Falconer. *Introduction to Quantitative Genetics*. Pearson Education India, 1996.

15 H.K. Finucane, B. Bulik-Sullivan, A. Gusev, G. Trynka, Y. Reshef, P-R. Loh, V. Anttila, H. Xu, C. Zang, K. Farh, et al. Partitioning heritability by functional annotation using genome-wide association summary statistics. *Nature Genetics*, 47(11):1228, 2015.

16 B. Friebe, J. Jiang, W.J. Raupp, R.A. McIntosh, and B.S. Gill. Characterization of wheat-alien translocations conferring resistance to diseases and pests: current status. *Euphytica*, 91(1):59–87, 1996.

17 M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, 1979.

**18** T. Ge, C-Y. Chen, Y. Ni, Y-C. Anne Feng, and J.W. Smoller. Polygenic prediction via Bayesian regression and continuous shrinkage priors. *Nature Communications*, 10(1):1–10, 2019.

**19** A.K. Gupta and D.K. Nagar. *Matrix Variate Distributions*. Chapman and Hall/CRC, 2018.

**20** T.F. Hansen. Stabilizing selection and the comparative analysis of adaptation. *Evolution*, 51(5):1341–1351, 1997.

**21** M. Jinek, K. Chylinski, I. Fonfara, M. Hauer, J.A. Doudna, and E. Charpentier. A programmable dual-rna–guided dna endonuclease in adaptive bacterial immunity. *Science*, 337(6096):816–821, 2012.

**22** E. Karavani, O. Zuk, D. Zeevi, G. Atzmon, N. Barzilai, N.C. Stefanis, A. Hatzimanolis, N. Smyrnis, D. Avramopoulos, L. Kruglyak, et al. Screening human embryos for polygenic traits has limited utility. *Cell*, 179(6):1424–1435, 2019.

**23** A.V. Khera, M. Chaffin, K.G. Aragam, M.E. Haas, C. Roselli, S.H. Choi, P. Natarajan, E.S. Lander, S.A. Lubitz, P.T. Ellinor, et al. Genome-wide polygenic scores for common diseases identify individuals with risk equivalent to monogenic mutations. *Nature Genetics*, 50(9):1219–1224, 2018.

**24** T.G. Kolda and B.W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, 2009. doi:10.1137/07070111X.

**25** S.A Lambert, L. Gil, S. Jupp, S.C. Ritchie, Y. Xu, A. Buniello, A. McMahon, G. Abraham, M. Chapman, H. Parkinson, et al. The polygenic score catalog as an open database for reproducibility and systematic evaluation. *Nature Genetics*, 53(4):420–425, 2021.

**26** T. Lencz, D. Backenroth, E. Granot-Hershkovitz, A. Green, K. Gettler, J.H. Cho, O. Weissbrod, O. Zuk, and S. Carmi. Utility of polygenic embryo screening for disease depends on the selection strategy. *Elife*, 10:e64716, 2021.

**27** T.H.E. Meuwissen, B.J. Hayes, and M.E. Goddard. Prediction of total genetic value using genome-wide dense marker maps. *Genetics*, 157(4):1819–1829, 2001.

**28** J. Pattee and W. Pan. Penalized regression and model selection methods for polygenic scores on summary statistics. *PLoS Computational Biology*, 16(10):e1008271, 2020. doi:10.1371/JOURNAL.PCBI.1008271.

**29** M. Paulis, A. Castelli, L. Susani, M. Lizier, I. Lagutina, M.L. Focarelli, C. Recordati, P. Uva, F. Faggioli, T. Neri, et al. Chromosome transplantation as a novel approach for correcting complex genomic disorders. *Oncotarget*, 6(34):35218–35230, 2015.

**30** M. Paulis, L. Susani, A. Castelli, T. Suzuki, T. Hara, L. Straniero, S. Duga, D. Strina, S. Mantero, E. Caldana, et al. Chromosome transplantation: A possible approach to treat human x-linked disorders. *Molecular Therapy-Methods & Clinical Development*, 17:369–377, 2020.

**31** J. Peter, M. De Chiara, A. Friedrich, J.-X. Yue, D. Pflieger, A. Bergström, A. Sigwalt, B. Barre, K. Freel, A. Llored, et al. Genome evolution across 1,011 *Saccharomyces cerevisiae* isolates. *Nature*, 556(7701):339–344, 2018.

**32** G. Petris, S. Grazioli, L. van Bijsterveldt, P. Murat, K.C. Liu, J. Birnbaum, J.E. Sale, and J.W. Chin. High-fidelity human chromosome transfer and elimination. *Science*, 390:1038–1043, 2025. doi:10.1126/science.adv9797.

**33** L.L. Qi, B. Friebe, P. Zhang, and B.S. Gill. Homoeologous recombination, chromosome engineering and crop improvement. *Chromosome Research*, 15(1):3–19, 2007.

**34** S.M. Richardson, L.A. Mitchell, G. Stracquadanio, K. Yang, J.S. Dymond, J.E. DiCarlo, D. Lee, C.L.V. Huang, S. Chandrasegaran, Y. Cai, J.D. Boeke, and J.S. Bader. Design of a synthetic yeast genome. *Science*, 355(6329):1040–1044, 2017.

**35** J.S. Sanjak, J. Sidorenko, M.R. Robinson, K.R. Thornton, and P.M. Visscher. Evidence of directional and stabilizing selection in contemporary humans. *Proceedings of the National Academy of Sciences*, 115(1):151–156, 2018.

**36** I.I. Schmalhausen. *Factors of evolution: the theory of stabilizing selection.* Blakiston, 1949.

37 V.A. Schneider, T. Graves-Lindsay, K. Howe, N. Bouk, H.-C. Chen, P.A. Kitts, et al. Evaluation of GRCh38 and de novo haploid genome assemblies demonstrates the enduring quality of the reference assembly. *Genome Research*, 27(5):849–864, 2017.

38 Y. Shen, G. Stracquadanio, Y. Wang, K. Yang, L.A. Mitchell, Y. Xue, Y. Cai, T. Chen, J.S. Dymond, K. Kang, et al. SCRaMbLE generates designed combinatorial stochastic diversity in synthetic chromosomes. *Genome Research*, 26(1):36–49, 2016.

39 N.Z. Shor. Quadratic optimization problems. *Soviet Journal of Computer and Systems Sciences*, 25(6):1–11, 1987.

40 P. M. Visscher, N. R. Wray, Q. Zhang, P. Sklar, M. I. McCarthy, M. A. Brown, and J. Yang. 10 years of GWAS discovery: Biology, function, and translation. *The American Journal of Human Genetics*, 101(1):5–22, 2017.

41 J. Yang, T.A. Manolio, L.R. Pasquale, E. Boerwinkle, N. Caporaso, J.M. Cunningham, M. De Andrade, B. Feenstra, E. Feingold, M.G. Hayes, et al. Genome partitioning of genetic variation for complex traits using common snps. *Nature Genetics*, 43(6):519, 2011.


# A Algorithms and Optimization Details

## A.1 Notations

We use a single *block-by-block* vectorization throughout the paper. For a matrix $X \in \mathbb{R}_{m\times n}$, $\mathrm{vec}(X) \in \mathbb{R}^{mn}$ is defined by $[\mathrm{vec}(X)]_{(i-1)n+j} = X_{ij}$ (stacking the rows $1,\ldots,m$ in order). Similarly, for a $3^{rd}$-order tensor $X \in \mathbb{R}_{m\times n\times p}$, $\mathrm{mat}(X) \in \mathbb{R}_{mn\times p}$ has row $(i-1)n+j$ equal to the fiber $X_{ij\bullet}$.

There are $2^T$ possible binary vectors of length $T$, with each such vector $\mathbf{d} \in \{0,1\}^T$, corresponding to an orthant $O_{\mathbf{d}} \subset \mathbb{R}^T$ defined as $O_{\mathbf{d}} \equiv \Big\{(x_1,..,x_T)\text{s.t.}(-1)^{d_i}x_i < 0, \forall i \in [T]\Big\}$. These $2^T$ orthants form a disjoint union of $\mathbb{R}^T$ (ignoring equalities with the axes).

In similar to eq. (1), for a vector $\mathbf{V} \in \mathbb{R}^p$, define the *3-mode* tensor-by-vector product as the matrix $[\boldsymbol{X} \times_3 \mathbf{V}] \in \mathbb{R}_{m\times n}$, with elements defined by:

$$[\boldsymbol{X} \times_3 \mathbf{V}]_{ij} \equiv \sum_{k=1}^{p} X_{ijk}V_k,\ \forall i \in [m], \forall j \in [n]. \tag{6}$$

**Element-wise notations.** For two vectors or matrices $A, B$ of the same size, their Hadamard product $A \odot B$ is the entrywise product (e.g. $[A \odot B]_{ij} = a_{ij}b_{ij}$ for matrices). For a matrix $A$, the column-wise maximum and minimum vectors are denoted $A_{\bigvee}, A_{\bigwedge}$, with $[A_{\bigvee}]_j = \max_i a_{ij}$ and $[A_{\bigwedge}]_j = \min_i a_{ij}$ (the max/min over rows, for each column).

## A.2 Proof of the pseudo-polynomial algorithm

**Proof of Proposition 5.** Shift each trait so that $\min_j X_{ijk} = 0$ for every $i, k$; this translates every achievable score vector, and the loss argument, by a fixed vector, leaving the minimizer unchanged. Every partial score vector then lies in the grid $\prod_{k=1}^{T}\{0,\ldots,R_k\}$. Let $S_i$ be the set of partial-sum vectors attainable using blocks $1,\ldots,i$: $S_0 = \{\mathbf{0}_T\}$ and $S_i = \{\boldsymbol{v} + X_{ij\bullet} : \boldsymbol{v} \in S_{i-1},\ j \in [C]\}$. Each $S_i$ is a subset of the grid, so $|S_i| \le \prod_k (R_k+1)$, and computing $S_1,\ldots,S_M$ costs $O(M\,C\,\prod_k R_k)$; returning $\min\{\mathcal{L}(\boldsymbol{v}) : \boldsymbol{v} \in S_M\}$ gives the exact optimum. As $\prod_k R_k$ is polynomial in the largest score magnitude for fixed $T$, this is a pseudo-polynomial algorithm. ◀

## A.3 Worst-case behavior of Branch-and-Bound

▶ **Proposition 6.** *In the worst case, the number of non-dominated vectors at stage $b$ of Algorithm 1 is $C^b$.*

**Proof.** We construct an instance with $T = 2$ traits as follows. Draw $u_{ij} \overset{i.i.d.}{\sim} U[0,1]$ for $i \in [M], j \in [C]$, and set $X_{ij\bullet} \equiv (u_{ij}, -u_{ij})$. For any selection $\mathbf{c} \in [C]^b$, the partial sum is $\sum_{i=1}^{b} X_{ic_i\bullet} = (s(\mathbf{c}), -s(\mathbf{c}))$, where $s(\mathbf{c}) \equiv \sum_{i=1}^{b} u_{ic_i}$. With probability 1 over the draws, the $C^b$ values $\{s(\mathbf{c}) : \mathbf{c} \in [C]^b\}$ are all distinct. For two selections $\mathbf{c} \neq \mathbf{c}'$ with $s(\mathbf{c}) < s(\mathbf{c}')$ the partial sums $(s(\mathbf{c}), -s(\mathbf{c}))$ and $(s(\mathbf{c}'), -s(\mathbf{c}'))$ are coordinatewise incomparable: the first has a smaller first coordinate but a larger second coordinate. Hence all $C^b$ partial sums are pairwise Pareto-incomparable and Algorithm 1 retains all of them at stage $b$. ◀

## A.4 The divide-and-conquer algorithm

Partition the $M$ chromosomes into $b$ disjoint sub-blocks $B_1, \ldots, B_b$ with $\uplus_i B_i = [M]$ and run Algorithm 1 on each sub-tensor $\boldsymbol{X}_{B_i\bullet\bullet}$ to obtain the Pareto-optimal set $\Gamma^{(i)}$ and sub-block optimum $\boldsymbol{X}^{(i)}_{\star}$. Writing $\boldsymbol{X}^{(i)}_{\textcircled{\vee}}$ ($\boldsymbol{X}^{(i)}_{\textcircled{\wedge}}$) for the coordinatewise max (min) over $\Gamma^{(i)}$, the global optimum satisfies $\mathcal{L}(\sum_i \boldsymbol{X}^{(i)}_{\textcircled{\vee}}) \le \mathcal{L}(\boldsymbol{X}_\star) \le \mathcal{L}(\sum_i \boldsymbol{X}^{(i)}_{\star}) \le \mathcal{L}(\sum_i \boldsymbol{X}^{(i)}_{\textcircled{\wedge}})$ for any monotonically non-increasing $\mathcal{L}$. The first inequality gives a lower-bound certificate; the second gives the best loss found so far, $\widehat{L} = \mathcal{L}(\sum_i \boldsymbol{X}^{(i)}_{\star})$, used for envelope pruning (line 7 of Algorithm 4). Algorithm 4 combines per-sub-block enumeration with Pareto- and envelope-pruning during the merge phase.

**Algorithm 4** Choosing Best Genome (Branch-and-Bound Divide-and-Conquer).

**Input:** $\boldsymbol{X}$, a $3^{rd}$-order tensor of polygenic scores.
**Parameters:** $\mathcal{L}$, a monotonically non-increasing loss; $\{B_1, \ldots, B_b\}$, a partition of $[M]$.

1: **for** $i \in [b]$ **do**
2: Run Algorithm 1 on $\boldsymbol{X}_{B_i\bullet\bullet}$ to get Pareto set $\Gamma^{(i)} = \{\boldsymbol{X}^{(i,k)} : k \in [S_i]\}$, sub-block optimum $\boldsymbol{X}^{(i)}_{\star}$, and envelopes $\boldsymbol{X}^{(i)}_{\textcircled{\vee}}, \boldsymbol{X}^{(i)}_{\textcircled{\wedge}}$.
3: $\widehat{L} \leftarrow \mathcal{L}\big(\sum_{i=1}^{b} \boldsymbol{X}^{(i)}_{\star}\big)$; $\mathcal{M}^{(0)} \leftarrow \{\emptyset\}$ with partial sum $\mathbf{0}_T$.
4: **for** $i = 1$ to $b$ **do**
5: MERGE: $\mathcal{M}^{(i)} \leftarrow \big\{(\mathbf{c}, \mathbf{c}^{(i,k)}) : \mathbf{c} \in \mathcal{M}^{(i-1)},\ k \in [S_i]\big\}$ with sums $P_{\mathbf{c}} + \boldsymbol{X}^{(i,k)}$.
6: PARETO-PRUNE: retain $\mathbf{c} \in \mathcal{M}^{(i)}$ only if $\nexists\, \tilde{\mathbf{c}} \in \mathcal{M}^{(i)}$ with $P_{\mathbf{c}} \prec P_{\tilde{\mathbf{c}}}$.
7: ENVELOPE-PRUNE: discard $\mathbf{c} \in \mathcal{M}^{(i)}$ if $\mathcal{L}\big(P_{\mathbf{c}} + \sum_{l=i+1}^{b} \boldsymbol{X}^{(l)}_{\textcircled{\vee}}\big) > \widehat{L}$.
8: Update $\widehat{L} \leftarrow \min_{\mathbf{c} \in \mathcal{M}^{(b)}} \mathcal{L}(P_{\mathbf{c}})$ and output $\mathbf{c}^\star = \arg\min_{\mathbf{c} \in \mathcal{M}^{(b)}} \mathcal{L}(P_{\mathbf{c}})$.

▶ Remark 7 (Correctness of Algorithm 4). For monotonically non-increasing $\mathcal{L}$ and any feasible extension $\mathbf{c}'$ of a partial selection $\mathbf{c}$ on the first $i$ sub-blocks, the partial sum satisfies $P_{\mathbf{c}'} \preceq P_{\mathbf{c}} + \sum_{l=i+1}^{b} \boldsymbol{X}^{(l)}_{\textcircled{\vee}}$ coordinatewise, hence $\mathcal{L}(P_{\mathbf{c}'}) \ge \mathcal{L}\big(P_{\mathbf{c}} + \sum_{l=i+1}^{b} \boldsymbol{X}^{(l)}_{\textcircled{\vee}}\big)$. If this lower bound exceeds the best loss found so far, $\widehat{L}$, no extension of $\mathbf{c}$ can improve on it, and the envelope-pruning step in Algorithm 4 is sound. Together with the Pareto-pruning step it follows that the final selection $\mathbf{c}^\star$ is globally optimal.

## A.5 Computing the Gradient

We show as an example the gradient computation for the stabilizing-selection loss and the threshold-linear (disease) loss. We also show that the simplex relaxation of Problem 1 (Section 3.4.1) is convex for the first case, and not convex for the second case.

1. For the stabilizing selection loss $\mathcal{L}(\boldsymbol{X}_{\mathbf{c}}) = \sum_i w_i \boldsymbol{X}_{\mathbf{c}_i}^2$, the relaxed problem is convex in $\mathbf{C}$ (the Hessian is a sum of rank-1 PSD outer products $\mathbf{x}_k \mathbf{x}_k^t$, $\mathbf{x}_k = \mathrm{vec}(\boldsymbol{X}_{\bullet\bullet k})$, weighted by $w_k > 0$), and its gradient and closed-form KKT solution are direct consequences of the quadratic form; the SDP relaxation of Section 3.4 provides a strictly tighter bound and is the route we follow in this paper.
2. For the threshold-linear loss $\mathcal{L} = \sum_i w_i P(D_i = 1|\boldsymbol{X}_{\mathbf{c}})$ with $P(D_i = 1|\boldsymbol{X}_{\mathbf{c}}) = \Phi\big((z_{K_i} - \boldsymbol{X}_{\mathbf{c}_i})/s_i\big)$ and residual standard deviations $s_i \equiv \sqrt{\boldsymbol{\Sigma}_{ii}^{(\boldsymbol{Y})} + \boldsymbol{\Sigma}_{ii}^{(\boldsymbol{e})}} = \sqrt{1 - \boldsymbol{\Sigma}_{ii}^{(\boldsymbol{X})}}$ (using the unit marginal-variance normalization), the gradient is

$$\frac{\partial \mathcal{L}(\mathbf{C})}{\partial C_{ij}} = -\sum_{k=1}^{T} \frac{w_k}{s_k}\, \phi\left(\frac{z_{K_k} - \boldsymbol{X}_{\mathbf{c}_k}}{s_k}\right) X_{ijk}, \tag{7}$$

with matrix form $\nabla\mathcal{L}(\mathbf{C}) = -\boldsymbol{X} \times_3 \Big[\mathbf{w} \odot \boldsymbol{s}^{-1} \odot \phi\big((\boldsymbol{z}_K - \boldsymbol{X}_{\mathbf{c}}) \odot \boldsymbol{s}^{-1}\big)\Big]$, where $\boldsymbol{z}_K = (z_{K_1}, \ldots, z_{K_T})$ and $\boldsymbol{s}_k^{-1} = 1/s_k$. Differentiating once more gives $\partial^2\mathcal{L}/\partial C_{ij}\partial C_{mc} = -\sum_k \beta_k X_{ijk} X_{mck}$ with $\beta_k \equiv w_k (z_{K_k} - \boldsymbol{X}_{\mathbf{c}_k})\, s_k^{-3}\, \phi\big((z_{K_k} - \boldsymbol{X}_{\mathbf{c}_k})/s_k\big)$, whose sign flips as $\boldsymbol{X}_{\mathbf{c}_k}$ crosses $z_{K_k}$; the Hessian is therefore not positive semidefinite uniformly over $\mathbf{C}$ and the loss is non-convex.

## B A Statistical Model from Quantitative Genetics

To put the abstract problem of the previous section in context, we describe a quantitative-genetics model for the joint distribution of the scores and non-score components determining a phenotype across $C$ genomes and $T$ complex traits. The model is general, applying to any organism for which polygenic scores can be computed, and it extends models used in [22, 26] for multiple traits, with two main differences: First, we model explicitly the joint distribution of the individual chromosomes' scores, whereas in [22, 26] a model was given for the entire genomic score. Second, [22, 26] considered embryos derived from the same two parents, yielding a specific genetic relationship matrix $\Sigma^{(C)}$ representing the Identity-by-Descent sharing of siblings, while in our case the genetic relationship matrix may be more general depending on the selection scenario (e.g. unrelated yeast strains or crop varieties, for which $\boldsymbol{\Sigma}^{(\boldsymbol{C})} \approx \boldsymbol{I}_{\boldsymbol{C}}$).

We assume that the genetic architecture of the traits is infinitesimal, namely that there are numerous causal variants, uniformly distributed along the genome. Denote the matrix of quantitative trait values as $\mathbf{Z} \in \mathbb{R}_{C\times T}$, where $Z_{ij}$ denotes the value of trait $j$ for the $i$-th copy. We can decompose $\mathbf{Z}$ as follows:

$$\mathbf{Z} = \sum_{i=1}^{M} \left(\boldsymbol{X}_{i\bullet\bullet} + \boldsymbol{Y}_{i\bullet\bullet}\right) + \boldsymbol{\epsilon} \tag{8}$$

where the error term $\boldsymbol{Y}$ represents a tensor of genetic components not accounted for by the scores $\boldsymbol{X}$, and the error term $\boldsymbol{\epsilon}$ represents a matrix of environmental components, both having zero mean.

We assume that all the traits have mean zero and variance 1, and further that the individual chromosome scores also have zero mean.

We further assume that the distribution of the polygenic scores $\boldsymbol{X}$ is approximately Normal in each genome (due to the polygenic nature of most complex traits [40]), and that the joint distribution of the polygenic scores over $C$ genomes is multivariate Gaussian.

Consider $T$ traits normalized to have zero mean and unit variance. Let $\boldsymbol{X}^{(e)} \equiv \sum_{i=1}^{M} \boldsymbol{X}_{i\bullet\bullet} \in \mathbb{R}_{C\times T}$ be the entire-genome score matrix for the $C$ copies, obtained by summing the per-chromosome score matrices (superscript $e$ for aggregation over all blocks, as in $G_e$; not the environmental $\boldsymbol{\Sigma^{(e)}}$), and similarly $\boldsymbol{Y}^{(e)} \equiv \sum_{i=1}^{M} \boldsymbol{Y}_{i\bullet\bullet} \in \mathbb{R}_{C\times T}$. The vector of polygenic score for a single genomic copy for all traits $\boldsymbol{X}_{i\bullet}^{(e)}$ has a covariance matrix $\boldsymbol{\Sigma^{(X)}}$ under a Normal model:

$$\boldsymbol{X}_{i\bullet}^{(e)} \sim MVN(\boldsymbol{0_T}, \boldsymbol{\Sigma^{(X)}}) \tag{9}$$

where $diag(\boldsymbol{\Sigma^{(X)}}) = (\sigma_{11}^{(X)}, .., \sigma_{TT}^{(X)})$ contains the variance explained by the polygenic scores of each trait, and the off-diagonal elements of $\boldsymbol{\Sigma^{(X)}}$ represent pleiotropic effects. For $C$ full-genome copies we obtain a $C \times T$ matrix of polygenic scores with a matrix Normal distribution [19]:

$$\boldsymbol{X}^{(e)} \sim MN_{C\times T}(\boldsymbol{0_{C\times T}}, \boldsymbol{\Sigma^{(C)}}, \boldsymbol{\Sigma^{(X)}}). \tag{10}$$

The matrix $\boldsymbol{\Sigma^{(C)}}$ represents (twice) the kinship coefficients between the $C$ full-genome copies. For example, when the copies represent unrelated strains or lines, $\boldsymbol{\Sigma^{(C)}} = \boldsymbol{I_C}$; when they represent sibling embryos, $\boldsymbol{\Sigma^{(C)}} = \frac{1}{2}[I_C + \boldsymbol{1}_C\boldsymbol{1}_C^t]$. We assume that the chromosome scores are independent, with the scores matrix of each chromosome having the matrix Normal distribution:

$$\boldsymbol{X}_{i\bullet\bullet} \sim MN_{C\times T}(\boldsymbol{0_{C\times T}}, \alpha_i^2\boldsymbol{\Sigma^{(C)}}, \boldsymbol{\Sigma^{(X)}}), \tag{11}$$

where $\alpha_i^2$ is the proportion of genetic variance explained by chromosome $i$. The genetic variances satisfy $\sum_{i=1}^{M} \alpha_i^2 = 1$, and this proportion is assumed to be the same for all traits, a consequence of the infinitesimal model and provided that the relative density of causal variants across the genome is similar across traits.

These contributions determine the utility of chromosomal selection, and are expected to be roughly proportional to chromosomes' length or to their number of genes. Here, we show a numerical analysis with $\alpha_i^2$ taken as the (normalized) chromosome lengths derived from the *S. cerevisiae* reference genome [8] and the human reference genome GRCh38 [37], shown in Table 2. The actual $\alpha_i^2$ may vary across traits and can be estimated per trait by heritability-partitioning methods [15, 41] when needed.

**Table 2** Relative chromosomal lengths (i.e. $\alpha_i^2$ values) for the yeast (16 chromosomes) and human (22 autosomes plus the X chromosome) reference genomes. The Y chromosome contributes negligibly to autosomal polygenic scores and is omitted. For yeast, $\sum_i \alpha_i \approx 3.89$; for the 23-block human set, $\sum_i \alpha_i \approx 4.68$.

**Yeast (*S. cerevisiae*, haploid, $M = 16$)**

| Chrom. | I | II | III | IV | V | VI | VII | VIII | IX | X | XI | XII | XIII | XIV | XV | XVI |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\alpha_i^2$ | .019 | .067 | .026 | .126 | .047 | .022 | .090 | .046 | .036 | .061 | .055 | .089 | .076 | .064 | .090 | .078 |

**Human (*H. sapiens*, diploid, $M = 23$)**

| Chrom. | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | X |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\alpha_i^2$ | .082 | .080 | .066 | .063 | .060 | .056 | .053 | .048 | .046 | .044 | .045 | .044 | .038 | .035 | .034 | .030 | .028 | .027 | .019 | .021 | .015 | .017 | .052 |

Similarly, the non-score genetic components are modeled as:

$$\begin{aligned}\boldsymbol{Y}^{(e)} &\sim MN_{C\times T}(\boldsymbol{0_{C\times T}}, \boldsymbol{\Sigma^{(C)}}, \boldsymbol{\Sigma^{(Y)}})\\ \boldsymbol{\epsilon} &\sim MN_{C\times T}(\boldsymbol{0_{C\times T}}, \boldsymbol{I_C}, \boldsymbol{\Sigma^{(e)}})\end{aligned} \tag{12}$$

where $\mathrm{diag}\big(\boldsymbol{\Sigma}^{(\boldsymbol{X})}+\boldsymbol{\Sigma}^{(\boldsymbol{Y})}+\boldsymbol{\Sigma}^{(\boldsymbol{e})}\big)=\mathbf{1}_T$, i.e. each trait is normalized to unit marginal variance while cross-trait covariances may be nonzero. The matrix $\boldsymbol{\Sigma}^{(\boldsymbol{X})}+\boldsymbol{\Sigma}^{(\boldsymbol{Y})}$ is known as the genetic covariance matrix, and can be estimated from GWAS data using e.g. methods like LD-Score-Regression [6]. The diagonal elements $h_i^2=\sigma_{ii}^{(X)}+\sigma_{ii}^{(Y)}$ are the narrow-sense heritabilities of the traits.

The matrix $\boldsymbol{\Sigma}^{(\boldsymbol{Y})}+\boldsymbol{\Sigma}^{(\boldsymbol{e})}$ is the covariance matrix of the residuals, and determines the conditional distribution of the phenotypes vector conditioned on the scores vector. For simplicity, our model makes several standard assumptions: no shared environment (hence the identity $\boldsymbol{I_C}$ is used as a covariance matrix for $\boldsymbol{\epsilon}$), and no assortative mating. If these assumptions are violated, this can be encoded by the covariance matrices of our model.

**Threshold traits.** For threshold (e.g. disease) traits we use the liability-threshold model [14]. Trait $k$ has prevalence $K_k\in[0,1]$ and liability $z_k=\boldsymbol{X}_{\mathbf{c}_k}+\varepsilon_k$, the selected score plus independent residual noise $\varepsilon_k$ of variance $s_k^2=\boldsymbol{\Sigma}_{kk}^{(\boldsymbol{Y})}+\boldsymbol{\Sigma}_{kk}^{(\boldsymbol{e})}=1-\boldsymbol{\Sigma}_{kk}^{(\boldsymbol{X})}$ (the liability variance not explained by the score, with $\boldsymbol{\Sigma}_{kk}^{(\boldsymbol{X})}$ the score's variance-explained on the liability scale). The status is the binary random variable $D_k=\mathbf{1}_{\{z_k<z_{K_k}\}}$ with threshold $z_{K_k}=\Phi^{-1}(K_k)$, and $\mathbf{D}=(D_1,\dots,D_T)$. Conditioning on the score,

$$P(D_k=1\mid\boldsymbol{X}_{\mathbf{c}})=\Phi\big((z_{K_k}-\boldsymbol{X}_{\mathbf{c}_k})/s_k\big),\tag{13}$$

so a large score lowers the crossing probability; the all-pass probability $P(\mathbf{D}=\mathbf{0}_T\mid\boldsymbol{X}_{\mathbf{c}})$ is the Gaussian orthant probability of $\{z_k\geq z_{K_k}\ \forall k\}$ under the residual covariance $\boldsymbol{\Sigma}^{(\boldsymbol{Y})}+\boldsymbol{\Sigma}^{(\boldsymbol{e})}$.

## C Implementation Details

Our algorithms and the simulation study presented in the paper are implemented in an R package available at `https://github.com/orzuk/EmbryoSelectionCalculator`, with the chromosomal-selection functions in `code/R/chrom`. Performance-critical routines are implemented in C++ via Rcpp [13].

# Reconciling and Comparing Variation Graphs Using Homology Relations

**Anna Lisiecka**
Institute of Informatics, University of Warsaw, Poland

**Adam Cicherski**
Institute of Informatics, University of Warsaw, Poland

**Norbert Dojer**
Institute of Informatics, University of Warsaw, Poland

## Abstract

In variation graphs genomic sequences are represented as paths sharing their nodes in homologous regions. Shared nodes must represent identical sequence fragments, but the definition does not impose strict criteria on when the paths should be merged and when not. Consequently, most variation graph building tools heuristically infer the graph structure from pairwise genome alignments or the results of seeding procedures of alignment algorithms.

In the current paper we introduce the concept of homology relation induced by a variation graph on the nucleotides of represented genomic sequences. This notion can be used to specify in a mathematically rigorous way the criteria the structure of a variation graph should meet. We investigate the relationships between the structures of variation graphs and their homology relations, as well as between operations on relations and on graphs. In particular, we show how to use the result of combining relations induced by different graphs to combine these graphs. Then, we propose homology-based methods of reconciling and comparing variation graphs representing the same genome collection. Moreover, we provide an implementation of homology-based tools for variation graph analysis.

**2012 ACM Subject Classification** Applied computing → Computational genomics

**Keywords and phrases** Pangenome, Variation Graph, Genome Homology

**Digital Object Identifier** 10.4230/LIPIcs.WABI.2026.29

**Supplementary Material** *Software*: `https://github.com/anialisiecka/vgrecon/` archived at `swh:1:dir:5c6c9a6e0bf9b74c40eb93f2b32b8beb452adfc6`

**Funding** This work was supported by the National Science Centre, Poland, under grant number 2022/47/B/ST6/03154.

## 1 Introduction

*Variation graphs* combine reference genomes and their variants into coherent data structures, representing genetic variation within a population. They intuitively represent both single nucleotide polymorphisms and large-scale structural variants (e.g. insertions, deletions and translocations). Efficient indexing methods [11, 13] make variation graphs an alternative to linear reference genomes that can reduce reference bias in mapping sequencing reads [7, 12].

Each variation graph node is labeled with a fragment of one or more individual genomes. Whole genomic sequences are represented as graph paths, and the concatenated labels of successive path nodes form the sequences. The framework is highly flexible in the sense that it does not impose constraints on when identical fragments of genomic sequences should be represented by a common node. Mathematically rigorous criteria the structure of a graph should meet were proposed in [2] and implemented in the variation graph building tool AlfaPang [3]. However, the authors note that AlfaPang may produce graphs with complex


© Anna Lisiecka, Adam Cicherski, and Norbert Dojer;
licensed under Creative Commons License CC-BY 4.0



local structures, and thus propose pipeline AlfaPang+, consisting of AlfaPang followed by the graph refinement tools smoothxg and gfafix [6]. Other variation graph building tools heuristically infer the graph structure from pairwise genome alignments or the results of seeding procedures of alignment algorithms.

It has been previously established [1] that variation graph construction tools exhibit structural instability, where identical input haplotypes may yield different graph representations across multiple runs. This instability may stem from the inherent non-determinism of the construction algorithms or the permutation of input sequences. For example, Minigraph-Cactus (MC) [9] requires the user to choose a reference genome that serves as the backbone of constructed graph. Other genomes are aligned to this backbone, and thus the similarity between non-reference sequence fragments may be overlooked if not accompanied by the similarity to the reference. As it was shown in [5], the choice of the reference genome has a significant impact on the topology of the resulting graph. PGGB [6] avoids the reference bias by inferring the graph structure from all-to-all pairwise genome alignments. However, its postprocessing procedures (used also in AlfaPang+) are non-deterministic, because the result depends on the order in which parallel computation threads complete their execution. Ensuring stability in graph construction is therefore essential to guarantee the reproducibility of various results and downstream analyses.

This raises the need to develop methods and tools for comparing and evaluating different graphs representing the same set of genomes. Usually such comparison is based on simple graph characteristics (size, node coverage by genomic paths, lengths of node labels, etc.), ignoring the way genomic sequences are represented within the graph. A better approximation of the similarity of the represented homology is given by the *segmentation distance* [5], derived from the sequence breakpoints induced by the edges of genomic paths. However, the segmentations are evaluated separately for each sequence, so this metric also indirectly compares the graph topology-encoded relationships between sequences.

In this article, we introduce the notion of a *homology relation*, which describes in a mathematically rigorous way how a variation graph models the sequence homology at the level of single nucleotides. The relation is representation-dependent, and the degree to which it reflects evolutionary homology in the classical sense depends on the particular graph construction method. The idea of introducing the concept of homology relations is inspired by the way of defining requirements for variation graphs in [2, 3], but its potential applications go beyond the formulation of graph properties. Here we investigate the relationships between the structures of variation graphs and their homology relations, as well as between operations on relations and on graphs. In particular, we show how to use the result of intersecting relations induced by different graphs to combine them into a single graph. Then, we propose homology-based methods of comparing variation graphs representing the same genome collection, specifically for evaluating a given graph against a gold standard, and for assessing the stability of graph construction algorithms. Moreover, we provide an implementation of homology-based tools for variation graph analysis, available at `https://github.com/anialisiecka/vgrecon/`.

## 2 Methods

### 2.1 Preliminaries

In bidirected graphs, every node has two sides (denoted here $\pm 1$), and undirected edges connect adjacent nodes on specific sides. More formally, the edge connecting side $o_1$ of node $v_1$ with side $o_2$ of node $v_2$ is an unordered pair $\{\langle v_1, o_1\rangle, \langle v_2, o_2\rangle\}$. A node $v$ can be

*oriented* in two possible ways, denoted $\langle v, +1\rangle$ and $\langle v, -1\rangle$. A *path* in a bidirected graph is a sequence of oriented nodes $\langle\langle v_1, o_1\rangle, \ldots, \langle v_m, o_m\rangle\rangle$ such that for all respective $j$, side $o_{j-1}$ of node $v_{j-1}$ is connected by an edge to side $-o_j$ of node $v_j$ (so a path enters and exits the node on opposite sides).

In variation graph the nodes are labeled with DNA sequences. The two possible node orientations correspond to the two strands of DNA molecules. The labeling function $l$ extends to oriented nodes: $l(\langle v, +1\rangle) = l(v)$ and $l(\langle v, -1\rangle)$ is the reverse complement of $l(v)$, and further to paths: $l(\langle\langle v_1, o_1\rangle, \ldots, \langle v_m, o_m\rangle\rangle) = l(\langle v_1, o_1\rangle) \ldots l(\langle v_m, o_m\rangle)$.

Consider a set of homologous sequences $S = \{s_1, \ldots, s_n\}$. A *variation graph representing* $S$ is defined as $G = \langle V, E, l, \pi\rangle$, where $\langle V, E, l\rangle$ is a labeled bidirected graph and $\pi$ is a mapping from $S$ to the set of $G$-paths satisfying the following conditions:

- $l(\pi(s_i)) = s_i$ for $i = 1, \ldots, n$,
- all the paths $\pi(s_i)$ jointly cover the whole graph $G$.

The paths $\pi(s_i)$ are called *genomic paths*.

Two representations $G = \langle V, E, l, \pi\rangle$ and $G' = \langle V', E', l', \pi'\rangle$ of $S$ are *isomorphic* if there exist a bijection $b$ between the sets of oriented nodes in $G$ and $G'$ satisfying the following conditions:

1. $b(\langle v, o\rangle) = \langle v', o'\rangle \iff b(\langle v, -o\rangle) = \langle v', -o'\rangle$,
2. $l(\langle v, o\rangle) = l(b(\langle v, o\rangle))$,
3. $\{\langle v_1, o_1\rangle, \langle v_2, o_2\rangle\} \in E \iff \{b(\langle v_1, o_1\rangle), b(\langle v_2, o_2\rangle)\} \in E'$,
4. $\pi(s_i) = \langle\langle v_1, o_1\rangle, \ldots, \langle v_m, o_m\rangle\rangle \iff \pi'(s_i) = \langle b(\langle v_1, o_1\rangle), \ldots, b(\langle v_m, o_m\rangle)\rangle$.

## 2.2 Homology relations

Below, we introduce the notion of the homology relation associated with a representation $G$ of $S$. Intuitively, this relation identifies the nucleotides in $S$-sequences that are merged along the paths representing them in the graph. The definition accounts for the fact that variation graphs can represent homology between both parallel and antiparallel DNA strands.

The set of *positions* in $S = \{s_1, \ldots, s_n\}$ is defined as $Pos(S) = \{\langle i, j, o\rangle \mid 1 \leq i \leq n \wedge 1 \leq j \leq |s_i| \wedge o = \pm 1\}$, where $|s_i|$ denotes the length of sequence $s_i$. The representation $G = \langle V, E, l, \pi\rangle$ decomposes $S$-sequences into the concatenation of the labels of their oriented nodes. In this decomposition, each nucleotide $s_i[j]$ corresponds to an occurrence of a specific nucleotide in $l(\langle v, o\rangle)$ for some $G$-node $v$ that occurs in $\pi(s_i)$ with orientation $o$. In this case, we say that the nucleotide from $l(\langle v, o\rangle)$ occurs in $S$ at position $\langle i, j, o\rangle$, and its complementary nucleotide from $l(\langle v, -o\rangle)$ occurs in $S$ at position $\langle i, j, -o\rangle$.

Two positions $\langle i, j, o\rangle, \langle i', j', o'\rangle \in Pos(S)$ are *homologous* with respect to $G$ if they correspond to occurrences of the same nucleotide in the label of an oriented node shared by the paths $\pi(s_i)$ and $\pi(s_{i'})$. The set of all pairs of positions homologous with respect to $G$ is called homology relation *associated* with $G$, and denoted by $h_G$ (see Figure 1).

It follows directly from the definition that $h_G$ is an equivalence relation on $Pos(S)$. Moreover, the equivalence classes of $h_G$ that group occurrences of complementary nucleotides in $l(\langle v, o\rangle)$ and $l(\langle v, -o\rangle)$, contain positions with inverse orientations, i.e.

$$[\langle i, j, -o\rangle]_{h_G} = \{\langle i', j', -o'\rangle \mid \langle i', j', o'\rangle \in [\langle i, j, o\rangle]_{h_G}\}. \tag{1}$$

It should be noted that the structure of the homology relation is strictly determined by the underlying variation graph representation. The degree to which it reflects homology in the evolutionary sense depends on the specific graph construction method.

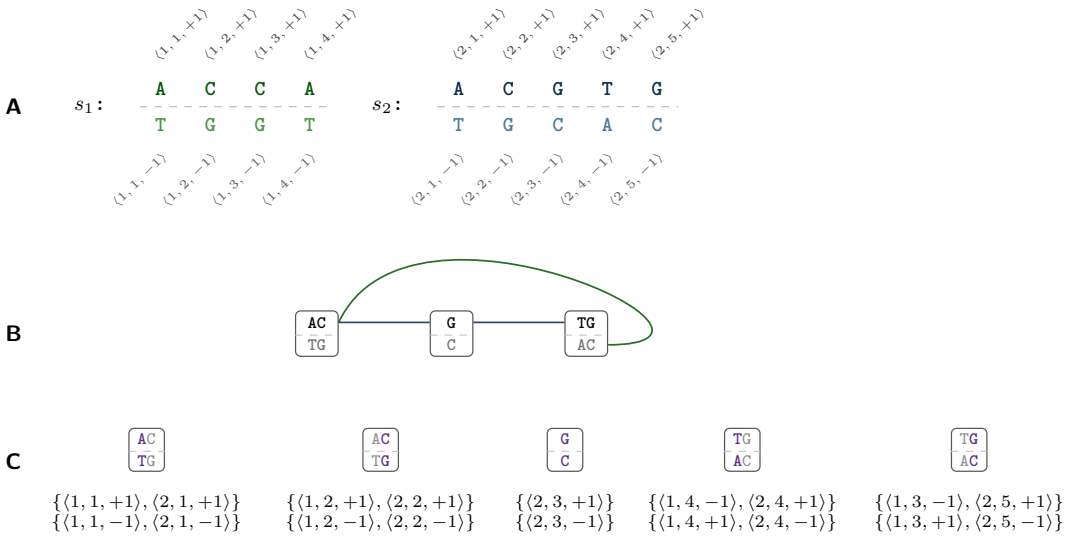



**Figure 1** Variation graph representation of two DNA sequences and the associated homology relation. (A) Sequence set $S = \{s_1, s_2\}$ presented alongside the elements of $Pos(S)$. Residues and positions on the $'+'$ strand are shown over the dashed lines, and those on the $'-'$ strand are shown under these lines. (B) Variation graph representing $S$. Consecutive nodes on $\pi(s_1)$ and on $\pi(s_2)$ are connected with green and blue edges, respectively. Sequences $s_1$ and $s_2$ are spelled by the labels of the nodes that are traversed by the respective paths from left to right (thus oriented $'+1'$), and the reverse complements of those traversed from right to left (thus oriented $'-1'$). (C) Equivalence classes of the homology relation induced by the graph shown in (B). Each pair of complementary nucleotides in each node label (highlighted in purple) induces two classes, grouping the positions, on which these nucleotides occur in $S$-sequences.

## 2.3 Equivalent and canonical graphs

We say that two representations $G$ and $G'$, of a sequence set $S$ are *equivalent* if they induce the same homology relation on $Pos(S)$. Isomorphic representations must be equivalent, but the opposite implication is not true. Below, we define structural modifications to the representation that do not alter the induced homology relations.

Given a path $\langle\langle v_1, o_1\rangle, \ldots, \langle v_m, o_m\rangle\rangle$, we refer to side $-o_1$ of node $v_1$ and side $o_m$ of node $v_m$ as *outer sides* of this path, while all other sides of its nodes are *inner* in this path. When all nodes in the path are different and each its inner side is incident with only one edge (connecting consecutive path nodes), the path is a *unitig*. A unitig having only one node is called *trivial*. *Splitting* a node $v$ consists of replacing it with a unitig having $|l(v)|$ nodes labeled with subsequent $l(v)$ characters. Splitting operation can be reversed by *compressing* a unitig into a single node, labeled with the concatenation of the labels of the unitig nodes. Unitig is called *safe*, if none of its inner sides is an outer side of a genomic path (in particular, unitigs created with the splitting operation are safe). It is easily seen that the operations of splitting a node and compressing a safe unitig result in an equivalent representation.

A representation is *singular* if all its nodes are labeled with single characters (and thus cannot be further split). A representation is *compact* if all its safe unitigs are trivial (and thus cannot be further compressed). Below, we prove that every equivalence class of variation graph representations contains both a singular and a compact representation, each possessing a uniquely determined structure.

▶ **Theorem 1.** *Equivalent singular variation graph representations of a sequence set $S$ are isomorphic.*

**Proof.** Assume $G$ and $G'$ are singular representations of $S$, inducing the same homology relation $h$ on $Pos(S)$. Due to singularity, each oriented $G$-node $\langle v, o\rangle$ is labeled with a single nucleotide, the S-occurrences of which form an equivalence class of h. This class must also group the positions of the $S$-occurrences of some oriented $G'$-node $\langle v', o'\rangle$. The above correspondence establishes the bijection $b$ defined as $b(\langle v, o\rangle) = \langle v', o'\rangle$ for all such pairs of oriented nodes in $G$ and $G'$.

We show that $b$ satisfies conditions 1-4 of the isomorphism definition. Conditions 1-2 follow directly from property (1) of homology relations. To show condition 4 let us denote oriented nodes representing position $\langle i, j, +1\rangle$ in $G$ and $G'$ by $\langle v_{i,j}, o_{i,j}\rangle$ and $\langle v'_{i,j}, o'_{i,j}\rangle$, respectively. The definition of $b$ implies that $b(\langle v_{i,j}, o_{i,j}\rangle) = \langle v_{i,j}, o_{i,j}\rangle$ for all respective $i$ and $j$. Moreover, each sequence $s_i \in S$ is represented by path $\pi(s_i) = \langle\langle v_{i,1}, o_{i,1}\rangle, \ldots, \langle v_{i,|s_i|}, o_{i,|s_i|}\rangle\rangle$ in $G$ and by path $\pi'(s_i) = \langle\langle v'_{i,1}, o'_{i,1}\rangle, \ldots, \langle v'_{i,|s_i|}, o'_{i,|s_i|}\rangle\rangle$ in $G'$. Combining the above observations yields condition 4. Finally, condition 3 is the consequence of the fact that the adjacency structure of a variation graph is induced by its genomic paths. Namely, for every $G$-edge $\{\langle v_1, o_1\rangle, \langle v_2, o_2\rangle\}$, either $\langle\langle v_1, o_1\rangle, \langle v_2, -o_2\rangle\rangle$ or $\langle\langle v_2, o_2\rangle, \langle v_1, -o_1\rangle\rangle$ is a pair of two consecutive oriented nodes in path $\pi(s_i)$ for some $i \in \{1, \ldots, n\}$. Consequently, either $\langle b(\langle v_1, o_1\rangle), b(\langle v_2, -o_2\rangle)\rangle$ or $\langle b(\langle v_2, o_2\rangle), b(\langle v_1, -o_1\rangle)\rangle$ is a pair of two consecutive oriented nodes in $\pi'(s_i)$, so $G'$ must contain edge $\{b(\langle v_1, o_1\rangle), b(\langle v_2, o_2\rangle)\}$. Similar argument applies to the opposite implications, which completes the proof. ◀

▶ **Theorem 2.** *Equivalent compact variation graph representations of a sequence collection are isomorphic.*

**Proof.** We first show that two different maximal safe unitigs cannot share a node. To this aim assume that two such unitigs share one. By definition, this node can be neither followed nor preceded by two different nodes within these unitigs. Since the unitigs are different, the terminal node of one unitig must be an internal node of the other one. Thus, the path forming the first unitig can be extended at this end with the next node from the other unitig. This contradicts the maximality of the first unitig, unless the node extending the path already belongs to it. However, in this case the unitig forms a maximal non-cyclic path on a cycle, in which every node is incident only to the edges of that cycle. This means that the cycle forms a separate connected component of the graph. Every node in this component must be covered by a genomic path, which therefore must terminate within it. Consequently, the cycle contains a terminal node of a genomic path, and thus an outer side of this path must be an inner side of at least one of the considered unitigs. This contradicts the assumption that both unitigs are safe.

The consequence of what we have shown is that two maximal safe unitigs that share a node are either identical or reversed paths. Therefore, the maximal safe unitigs partition the set of graph nodes into subsets that can be compressed independently. Thus every compact representation is the result of compressing all maximal safe unitigs in an equivalent singular representation. When two compact representations are equivalent, the singular representations are also equivalent, and hence isomorphic. As a result, equivalent compact representations are isomorphic too. ◀

## 2.4 Graph comparison

Different variation graphs representing the same sequence set are usually compared based on graph structure characteristics, e.g. the number of nodes, the length of node labels, or the number of genomic paths covering the node. In general, such measures poorly reflect the similarity between the homologies of represented sequences; graphs with very similar

structural properties can have completely different homology relations, while equivalent singular and compact representations can significantly differ in the size of their graphs and the length of node labels. A better approximation of the similarity between homologies is given by the *segmentation distance* [5]. Each edge on a genomic path establishes a *breakpoint* between two adjacent positions in the represented sequence. The set of all breakpoints on a genomic sequence is its *segmentation*. The segmentation distance is the size of symmetric difference between the segmentations induced by the representations. The representations are assumed to be compact, so the edges separate regions of homogeneous homology relations, and the segmentation distance quantifies the differences between the projections of these regions on the genomic sequences. Although projections capture homology more accurately than structural graph characteristics, the segmentation distance may still distort the representation of similarity between homologies. This effect is illustrated in Figure 2, where the distance between Graph B and Graph C is larger than between Graph A and Graph B, despite the latter pair differing more substantially in their encoded homology relations. This indicates that segmentation distance remains more sensitive to changes in segmentation and graph topology than to the underlying homology structure itself.

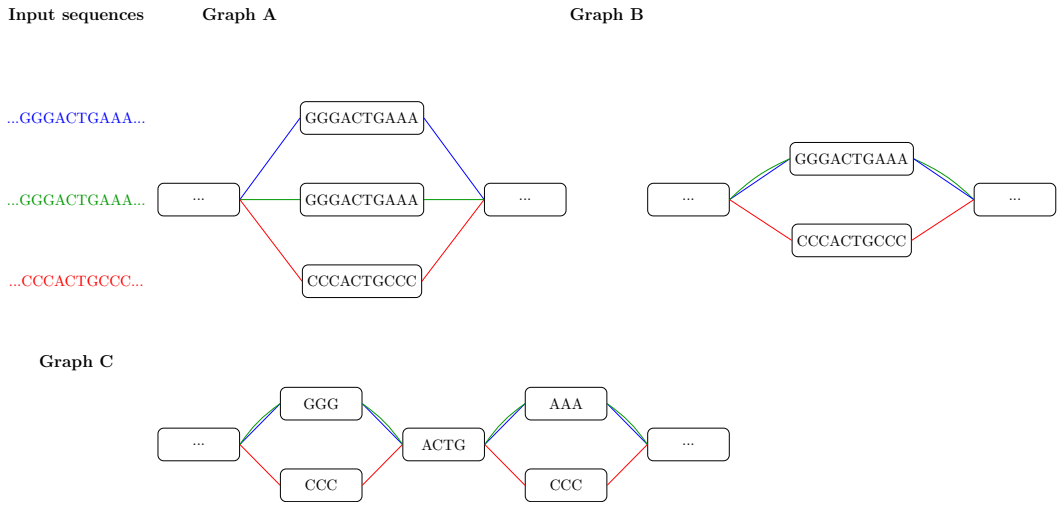



**Figure 2** An example illustrating limitations of segmentation distance. The figure shows different subgraphs representing fragments of three DNA sequences. Dots indicate that the displayed subgraphs continue beyond the illustrated region. Multi-edges are used only for visualization purposes, and edge colors correspond to the respective genomic paths. In Graph A, each sequence fragment is represented by a separate node. In Graph B, the identical blue and green sequences are represented by a common node. In Graph C, the shared segment `ACTG`, present in all three sequences, is merged into single. The blue and green sequences are represented by the identical path. Graphs A and B have segmentation distance equal to 0, despite representing substantially different homology relationships. In contrast, the segmentation distance between Graphs B and C equals 6, even though Graph C preserves most of the homology structure represented in Graph B. In both graphs, the two identical sequence fragments remain recognized as homologous, while Graph C additionally merges the shared segment `ACTG` of the third sequence.

The limitations of the above approaches to evaluating variation graph similarity can be overcome by directly comparing the underlying homology relations. This comparison can be performed using any standard set similarity measure, e.g. the Jaccard distance. We apply this distance to the relations $\widehat{h_G} = h_G \setminus Id_{Pos(S)}$, where $Id_{Pos(S)}$ is the identity relation on

the set of all positions in $S$. The reason for this is that pairs of identical positions belong to every homology relation on $Pos(S)$. Thus, the *homology distance* between two homology relations is defined by the following formula:

$$d(h_1, h_2) = 1 - \frac{|\widehat{h_1} \cap \widehat{h_2}|}{|\widehat{h_1} \cup \widehat{h_2}|}$$

and between two variation graphs by $d_h(G_1, G_2) = d(h_{G_1}, h_{G_2})$.

▶ **Proposition 3.** *Distance function $d$ is a metric on the set of homology relations on $Pos(S)$.*

**Proof.** Since $d$ is defined via the Jaccard distance between finite sets of non-trivial homology pairs, it directly inherits the metric properties of the Jaccard distance over finite sets. ◀

The distance function $d_h$ satisfies the metric properties except the identity axiom, since $d_h(G_1, G_2) = 0$ whenever $G_1$ and $G_2$ are equivalent. However, both $d$ and $d_h$ satisfy the triangle inequality, unlike most heuristic similarity scores.

## 2.5 Graph reconciliation

Consider two representations $G_1$ and $G_2$ of a sequence set $S$. Representation $G$ of $S$ is the *reconciliation* of $G_1$ and $G_2$ if $h_G = h_{G_1} \cap h_{G_2}$.

▶ **Theorem 4.** *The singular (respectively compact) reconciliation of representations $G_1$ and $G_2$ of the same sequence set $S$ always exists and is unique up to isomorphism.*

**Proof.** First we will show the construction of a singular $S$-representation with homology relation $h = h_{G_1} \cap h_{G_2}$. Since both $h_{G_1}$ and $h_{G_2}$ are equivalence relations, $h$ is an equivalence relation, too. A similar argument leads to the observation that the equivalence classes $[\langle i, j, +1\rangle]_h$ and $[\langle i, j, -1\rangle]_h$ group positions with the same coordinates $i, j$ and inverse orientations.

A singular representation $G$ of $S$ is constructed as follows:

- each node of $G$ represents a pair of $h$-classes grouping positions with inverse orientations,
- the two classes within the pair correspond to the two possible node orientations,
- consequently, every $h$-class $c$ is represented by an oriented node, denoted below by $vo(c)$,
- $l(vo([\langle i, j, +1\rangle]_h)) = s_i[j]$ and $l(vo([\langle i, j, -1\rangle]_h))$ is the complement of $s_i[j]$,
- $\pi(s_i) = \langle vo([\langle i, 1, +1\rangle]_h), \ldots, vo([\langle i, |s_i|, +1\rangle]_h)\rangle$ for $i = 1, \ldots, n$,
- the edges of $G$ connect nodes that are adjacent along the paths $\pi(s_i)$, respecting their orientations.

Resulting singular representation can be transformed into a compact one by compressing its maximal safe unitigs into single nodes, as in the proof of Theorem 2.

Finally, the uniqueness of singular and compact representations is a direct consequence of their existence and Theorems 1 and 2. ◀

Reconciliation can be applied to variation graph evaluation in several ways. First, it simplifies the problem of homology distance calculation – once we compute the sizes of homology relations induced by $G_1$, $G_2$ and their reconciliation $G$, their homology distance is given by formula

$$d_h(G_1, G_2) = 1 - \frac{|\widehat{h_G}|}{|\widehat{h_{G_1}}| + |\widehat{h_{G_2}}| - |\widehat{h_G}|}.$$

Second, the ratios $|\widehat{h_G}|/|\widehat{h_{G_1}}|$ and $|\widehat{h_G}|/|\widehat{h_{G_2}}|$ serve as asymmetric similarity measures, describing the proportion of position pairs in one relation that also belong to the other. We will call each of these ratios the *homology reconciliation index (HRI)*, and denote by $HRI_1(G_1, G_2)$ and $HRI_2(G_1, G_2)$, respectively. When one of the representations is evaluated with respect to the other (serving as a *gold standard*), the ratios establish the statistics *precision* and *recall*.

Finally, we generalize the concept of reconciliation to any number of input representations: $G$ is a reconciliation of representations $G_1, \ldots, G_n$ if $h_G = h_{G_1} \cap \ldots \cap h_{G_n}$. It can be computed by iteratively applying the above construction, and the result does not depend on the order of input representations. The notion of HRI is generalized accordingly:

$$HRI_i(G_1, \ldots, G_n) = |\widehat{h_G}|/|\widehat{h_{G_i}}|.$$

When applied to representations constructed in different runs of the same graph building tool, HRIs can be used to quantify the stability of its construction algorithm. Their values range from 0 to 1, where HRI of exactly 1 indicates that all homology pairs from the individual run are perfectly preserved within the reconciled graph. Conversely, lower values indicate that the individual graph instance induces a broader homology relation, containing pairs that are not universally shared across the other runs.

## 2.6 Computing reconciliation graph and similarity measures

All we need to calculate the above mentioned measures is to compute the size of relation $\widehat{h_G}$ for a given graph $G = \langle V, E, l, \pi \rangle$ and to construct the reconciliation of two given graphs $G_1$ and $G_2$.

The first task is straightforward. It is easily seen that a node $v \in V$ that has $k_v$ occurrences in genomic paths of graph $G$, induces $2|l(v)|$ equivalence classes of relation $h_G$, and each of these classes consists of $k_v$ positions. Moreover, an $h_G$-class with $k$ elements contributes $k^2$ pairs to relation $h_G$, and thus $k(k-1)$ pairs to relation $\widehat{h_G}$. Consequently, the size of $\widehat{h_G}$ is given by formula

$$|\widehat{h_G}| = \sum_{v \in V} 2|l(v)|k_v(k_v - 1).$$

Therefore, $|\widehat{h_G}|$ can be computed in linear time with respect to the size of graph $G$.

A naive approach to the second task (by enumerating all pairs in $\widehat{h_{G_1}}$ and $\widehat{h_{G_2}}$, intersecting the results, and building the singular graph as in the proof of Theorem 4) leads to an algorithm with $O(k_{max}^2 N)$ time and space complexity, where $k_{max} = \max_{v \in V} k_v$ and $N$ is the total length of genomic sequences. This is too much when it comes to processing large genomic data sets.

Below we propose a reconciliation construction algorithm, whose complexity is independent on both parameters. Given variation graph $G = \langle V, E, l, \pi \rangle$, we define $P_G = \sum_{v \in V} k_v$. Note that all components of $G$ except $l$ can be encoded in space $O(P_G)$. Our algorithm reconciles variation graphs $G_1$ and $G_2$ in time and space depending on parameters $P_{G_1}$ and $P_{G_2}$ only. To shorten the notation, we denote these parameters below as $P_1$ and $P_2$, respectively.

A node $v$ in a reconciliation graph $G$ is *trivial* if $k_v = 1$. Because trivial nodes induce only singleton equivalence classes of $h_G$, they contribute no pairs to $\widehat{h_G}$. Therefore, they are redundant for the computation of $|\widehat{h_G}|$, so our algorithm omits them from the output reconciliation graph. When a complete reconciliation graph is needed, the output can be postprocessed using the GFALace tool [8], which recovers trivial nodes and reconstructs genomic paths.

In our algorithm the segmentation induced by $G_1$ serves as a reference. Its breakpoints are sorted according to their order on the genomic sequences, and the overlaps between a query sequence fragment and the reference segments can be computed via binary search. The nodes of $G_2$ are processed consecutively and their occurrences are queried in this structure. Processing a node $v$ yields non-trivial nodes representing those sequence positions in the reconciliation graph that were represented by $v$ in $G_2$. This is done in the following steps:

1. **Breakpoint detection.** For each occurrence of $v$, all $G_1$-breakpoints located within this occurrence are identified, along with respective sides of the two $G_1$-nodes, whose occurrences the breakpoint separates. Moreover, for each breakpoint the corresponding position in $l(v)$ is calculated.
2. **Sorting and grouping.** Breakpoints identified within all occurrences are sorted together according to their $l(v)$-positions and split into groups sharing this position.
3. **Creating reconciliation nodes.** Each group of breakpoints is further divided into subgroups sharing the same $G_1$-node and its side. The division is performed twice, with respect to $G_1$-nodes representing fragments starting and ending at the $l(v)$-position. Subgroups assigned to opposite sides of the same $G_1$-node are called *matching*. Then three types of reconciliation nodes can be created:
   - $v$ occurrences with no $G_1$-breakpoint inside form a single node,
   - each pair of matching subgroups yields a node, whose occurrences form a subset of the occurrences of respective $G_1$-node,
   - each non-matching subgroup yields a node, whose occurrences have one end inherited from the respective $G_1$-node and its side, and the other end inherited from one of sides of $v$.

An example illustrating the reconciliation algorithm is presented in Figure 3. Below we analyze its computational complexity.

▶ **Theorem 5.** *The time complexity of the reconciliation algorithm is* $O((P_2 + P_1) \log P_1)$.

**Proof.** The construction of the reference graph structure is dominated by sorting its breakpoints, which requires $O(P_1 \log P_1)$ time. Below we analyze separately the cost of each step of processing a $G_2$-node, summed up over all $G_2$-nodes.

1. The algorithm iterates through all $P_2$ genomic occurrences of $G_2$-nodes. For each occurrence, the algorithm performs the binary search over the sorted vector of $G_1$-breakpoints to locate the occurrence starting position. This requires $O(\log P_1)$ time, so the total time of all searches is $O(P_2 \log P_1)$. Each search is followed by enumerating $G_1$-breakpoints located within the occurrence. Since each $G_1$-breakpoint belongs to only one $G_2$-node occurrence, the total time of the enumeration phase is $O(P_1)$. Thus the total time of step 1 is $O(P_1 + P_2 \log P_1)$.
2. Denote the number of $G_1$-breakpoints that are located within the occurrences of a $G_2$-node $v$ by $P_v$. Then sorting these breakpoints needs $O(P_v \log P_v)$ time, and sorting the breakpoints located within the occurrences of all $G_2$-nodes needs time
$$O(\sum_{v \in V_2} P_v \log P_v) = O(\sum_{v \in V_2} P_v \log P_1) = O(P_1 \log P_1).$$
3. Using the lookup tables, the whole step for a $G_2$-node $v$ can be executed in a single iteration over the sorted set of the sides of $G_1$-breakpoints located within $v$-occurrences, enhanced with the ends of these occurrences. This needs $O(P_v + k_v)$ time, resulting in $O(\sum_{v \in V_2}(P_v + k_v)) = O(P_1 + P_2)$ time for all $G_2$-nodes.

Combining the above results we obtain the total time complexity $O((P_2 + P_1) \log P_1)$, which completes the proof. ◀

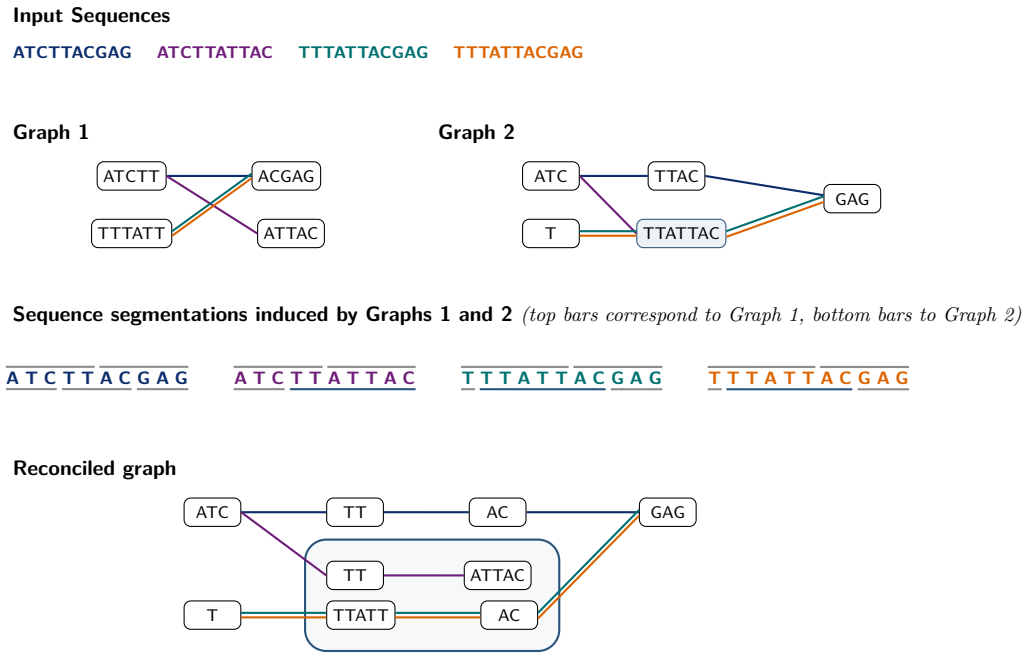



**Figure 3** Workflow of the variation graph reconciliation algorithm. The figure illustrates the input sequences, their individual graph structures (Graph 1 and Graph 2), input sequence segmentations induced by both graphs (top bars correspond to segmentation induced by Graph 1, bottom bars to Graph 2, with the segments corresponding to the highlighted node from Graph 2 marked in blue), and the final output as a reconciled graph, where the blue rectangle indicates how the shaded node from Graph 2 has been partitioned in the reconciled graph.

▶ **Theorem 6.** *The space complexity of the reconciliation algorithm is* $O(P_1 + P_2)$.

**Proof.** First, the segmentation data for the reference graph $G_1$ is stored using parallel vectors – specifically dedicated to the starting positions, corresponding node IDs, and orientations of each interval – requiring $O(P_1)$ space. Similarly, the grouped interval structures for the query graph $G_2$ require $O(P_2)$ space.

Second, processing a $G_2$-node $v$ requires local structures for storing ends of $v$-occurrences and $G_1$-breakpoints located within these occurrences. The size of these structures is $O(P_v+k_v)$ (using the notation from the proof of Theorem 5). Thus all the structures used in the algorithm have their size bounded by $O(P_1 + P_2)$, which completes the proof. ◀

The C++ implementations of the reconciliation algorithm and of the $|\widehat{h_G}|$ calculator are available at `https://github.com/anialisiecka/vgrecon/`.

## 3 Experiments

We conducted several series of experiments. First, we investigated the structural stability of variation graph construction tools across multiple runs. Second, we compared homology and segmentation distances calculated for variation graphs built using different tools. Third, we evaluated the scalability of our graph reconciliation algorithm by varying the number of genomes in the input dataset.

To evaluate the stability of variation graph construction tools, we generated 10 graph instances for Minigraph-Cactus, PGGB, AlfaPang and AlfaPang with graph simplification steps of PGGB (namely `smoothxg`), which we denote AlfaPang+ and measured their homology reconciliation index. The datasets were constructed using 10 human haplotypes

of chromosome 6. For Minigraph-Cactus, we varied the reference sequence across the 10 runs to analyze how the choice of reference affects the final graph structure. For PGGB, we executed the tool 10 times with identical parameters to evaluate the impact of its inherent non-determinism. For AlfaPang and AlfaPang+, identical configurations were used across all 10 runs to evaluate structural consistency under alignment-free and post-processed workflows. For AlfaPang+, the analysis evaluated two distinct graph refinement configurations using `smoothxg`: the first variant used its native default parameters, while the second variant applied the default parameters native to PGGB.

The analysis of the homology reconciliation index (see Table 1) reveals significant differences in the structural stability of the graphs generated by these tools. For Minigraph-Cactus, a relatively low degree of similarity between instances was observed, with a mean similarity of 0.1 and values spanning from 0.06 to 0.19. This is a direct consequence of the algorithm's sensitivity to the choice of the reference sequence – changing the reference during construction significantly alters the final graph topology and node segmentation, resulting in divergent homology representations.

In contrast, PGGB exhibits a markedly different characteristic. Despite its inherent non-determinism, the generated graphs show an extremely high level of structural consistency relative to the reconciled graph, maintaining HRI between 0.9961 and 0.9964. Such stable values indicate that the graphs produced by PGGB remain nearly identical across all trials, with minor fluctuations having only a marginal impact on the final genomic representation.

The next tool, AlfaPang, has a fully deterministic algorithm, assuring perfect reproducibility across different runs, which is reflected by HRI constantly equal to 1.

In contrast, the inclusion of a graph refinement phase in the AlfaPang+ workflow introduces notable structural changes, the extent of which depends heavily on the chosen post-processing parameters. When using the native default settings of `smoothxg`, the mean HRI decreases to 0.65, with individual values ranging from 0.62 to 0.66. However, when AlfaPang+ is configured with the PGGB-native parameters for `smoothxg`, the structural consistency improves drastically. This variant achieves a very high mean HRI of 0.9915, closely matching the performance of PGGB itself, with values tightly bounded between 0.9914 and 0.9917. This comparison highlights that the choice of `smoothxg` parameters affects not only the final graph structure, but also the reproducibility of the graphs across independent runs.

**Table 1** Summary of the homology reconciliation index between input variation graphs and their respective reconciliations. The analysis covers the mean, standard deviation, minimum, and maximum values across 10 runs of each tool. For Minigraph-Cactus, the variation originates from using different reference sequences during graph construction. For PGGB, the differences are a result of the algorithm's inherent non-determinism. For AlfaPang, the zero variance reflects its deterministic and alignment-free nature, while for AlfaPang+, the fluctuations arise from the subsequent graph refinement phase.

| Statistic | Minigraph-Cactus | PGGB | AlfaPang | AlfaPang+[1] | AlfaPang+[2] |
|---|---|---|---|---|---|
| Mean | 0.1007 | 0.9963 | 1 | 0.6474 | 0.9915 |
| SD | 0.0424 | 0.0001 | 0 | 0.0134 | 0.0001 |
| Min | 0.0567 | 0.9961 | 1 | 0.6197 | 0.9914 |
| Max | 0.1866 | 0.9964 | 1 | 0.6621 | 0.9917 |

1 Refinement step performed using `smoothxg` with its native default parameters.
2 Refinement step performed using `smoothxg` with default parameters from PGGB.

For the second experiment, we used simulated bacterial genomes previously employed for the evaluation of SibeliaZ [10]. The datasets were generated using ALF [4], a tool designed to simulate bacterial genome evolution. The evolutionary model of ALF incorporates both point mutations (insertions, deletions, and substitutions) and large-scale structural variations, including translocations and gene gain or loss events. We selected six distinct datasets simulated at varying phylogenetic distances from their common ancestor, with divergence levels ranging from 0.03 to 0.18 substitutions per site. Each dataset comprises 10 bacterial genomes, approximately 1.5 Mbp in size and containing 1500 genes. The raw genomic sequences from each dataset served as direct input to four distinct variation graph construction tools: PanGenome Graph Builder (PGGB) [6], Minigraph-Cactus [9], AlfaPang and AlfaPang+ [3].

All these tools were applied to genomic sequences generated by ALF and the output variation graphs were compared using two metrics: the homology distance and the normalized version of the segmentation distance. The original segmentation distance was defined in [5] as the number of sequence breakpoints induced by only one of the two graphs. We divide this number by the total number of breakpoints induced by both graphs, which ensures that the distance falls in the range [0,1], and thus it is directly comparable with the homology distance. Figure 4 summarizes the results. For each pair of variation graph-building tools, both metrics follow the same trend – the distances increase with increasing divergence between input sequences. Moreover, both metrics have very similar values for the comparison of graphs constructed using PGGB, Minigraph-Cactus and AlfaPang+. On the other hand, when comparing a graph created using AlfaPang with a graph from another tool, the segmentation distance is much larger than the homology distance, especially for datasets with low genomic divergence. This effect arises, because, without additional graph simplification, AlfaPang

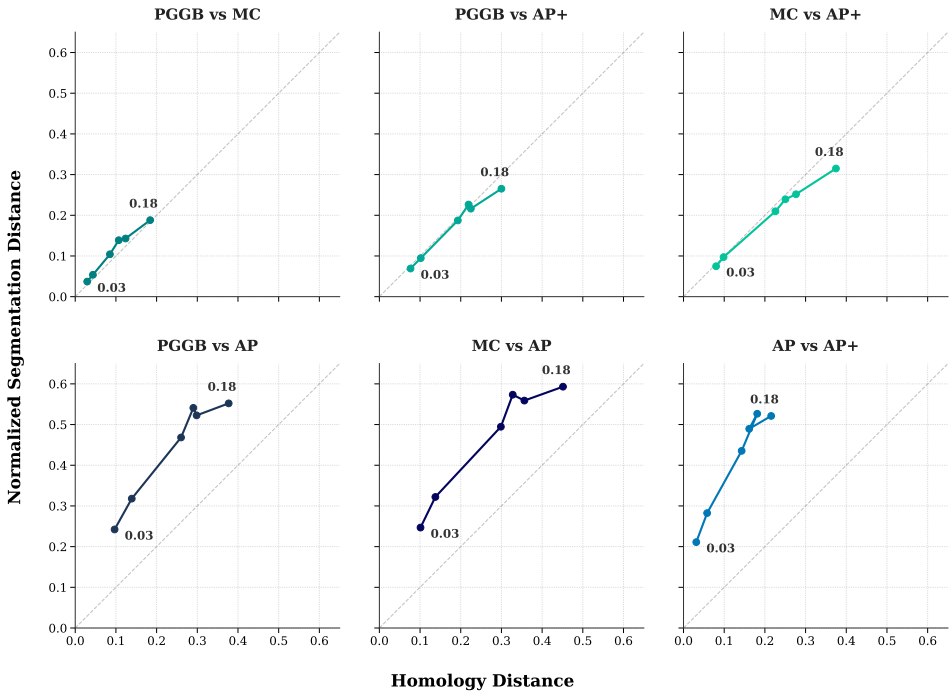



**Figure 4** Distances between different variation graphs. Numerical labels correspond to the genomic divergence of the simulated dataset. MC: Minigraph-Cactus, AP: AlfaPang, AP+: AlfaPang+.

may merge relatively short sequence fragments into single nodes. As a consequence, longer homologous sequences can become fragmented into multiple vertices, similarly to the effect illustrated in the Figure 2. Furthermore, while this fragmentation creates numerous artificial breakpoints – which drastically increases the segmentation distance – these merged fragments are of limited length. Consequently, they have a negligible impact on the total homology relation, leaving the homology distance largely unchanged. This highlights a major advantage of our metric: while structural distances fluctuate drastically based on variations in sequence partitioning, the homology distance scales with the lengths of sequence fragments with modified homology.

To evaluate the scalability of our tool, we conducted experiments on datasets consisting of 50, 100, 200, and 400 *Escherichia coli* genomes, allowing us to assess performance across varying input sizes. For each dataset, two variation graphs were generated using PGGB and Minigraph-Cactus, respectively, and used as an input for the reconciliation construction algorithm. Performance results (see Figure 5) illustrate the resource requirements for the core reconciliation algorithm and the postprocessing step, performed with the GFALace tool. The evaluation shows that our algorithm scales efficiently with the number of genomes. This reflects the efficiency achieved by operating at the graph topology level rather than processing raw, redundant sequence lengths. Surprisingly, the postprocessing step introduces much larger computational footprint – it is $\sim 4\times$ slower and requires $\sim 2\times$ more RAM.

## 4 Conclusions and discussion

We introduced the notion of homology relation, which describes in a mathematically rigorous way how a variation graph models the sequence homology at the level of single nucleotides. We analyzed the structure of classes of variation graphs sharing the same homology relation and characterized their canonical representatives. Furthermore, we used this concept to define the notion of a variation graph that reconciles two or more graphs, and developed an efficient algorithm for its construction.

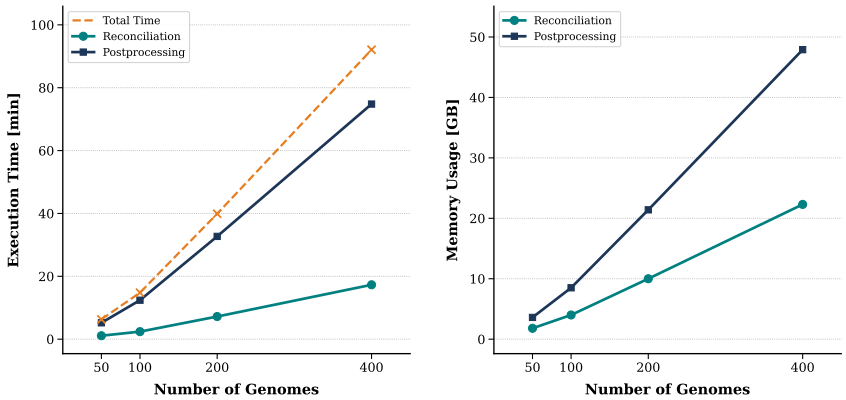



**Figure 5** Computational resource consumption analysis across different pangenome sizes. Left panel illustrates the execution time for the reconciliation phase and the postprocessing phase, as well as the total execution time. Right panel depicts memory usage across both phases. The experiments were performed on variation graphs constructed using PGGB and Minigraph-Cactus pipelines, from 50, 100, 200, and 400 *E. coli* genomes. The computations were performed on a Supermicro X10DRi server with 512GB RAM and two 14-cores CPUs Intel Xeon E5-2690V4, using a single thread.

Moreover, we proposed two homology-based measures that can be used to evaluate variation graphs. The first one, homology distance, is a metric for comparing graphs representing the same genome collection. Unlike heuristic similarity scores based on graph structural features, homology distance directly compares represented homologies while satisfying the axioms of metric. The second measure, homology reconciliation index, can be used to evaluate in more detail a given graph against the true one, and to assess the stability of variation graph construction algorithms. In particular, it quantifies the extent to which a homologous relation is consistently reproduced across different runs of the same pipeline, and the extent to which it is run-specific. In this way, it can support the construction of reference pangenome graphs by characterizing the impact of inherent non-determinism in graph building tools or of their parameter settings.

We implemented efficient algorithms to compute the above measures and made the software available at `https://github.com/anialisiecka/vgrecon/`. Finally, we verified the applicability of our contribution in a series of experiments.

To the best of our knowledge, the reconciliation operation proposed in this work is the first method to combine multiple variation graphs representing the same set of sequences into a single graph. It is based on the paradigm that the resulting graph should contain only homology relationships that are consistently accepted by all input graphs. Although fully justified in the above evaluation schemes, in the context of the graph combining problem this paradigm may be considered overly conservative, because it can eliminate biologically meaningful homologies missed in single input graphs. The development of less conservative approaches (e.g. union- or consensus-based) is a subject of future work.

## References


1 Francesco Andreace, Pierre Lechat, Yoann Dufresne, and Rayan Chikhi. Comparing methods for constructing and representing human pangenome graphs. *Genome Biology*, 24(1), November 2023. doi:10.1186/s13059-023-03098-2.

2 Adam Cicherski and Norbert Dojer. From de bruijn graphs to variation graphs – relationships between pangenome models. In Franco Maria Nardini, Nadia Pisanti, and Rossano Venturini, editors, *String Processing and Information Retrieval 2023*, pages 114–128, Cham, 2023. Springer Nature Switzerland. doi:10.1007/978-3-031-43980-3_10.

3 Adam Cicherski, Anna Lisiecka, and Norbert Dojer. Alfapang: alignment free algorithm for pangenome graph construction. *Algorithms Mol. Biol.*, 20(1):7, 2025. doi:10.1186/S13015-025-00277-7.

4 Daniel A. Dalquen, Maria Anisimova, Gaston H. Gonnet, and Christophe Dessimoz. Alf—a simulation framework for genome evolution. *Molecular Biology and Evolution*, 29(4):1115–1123, December 2011. doi:10.1093/molbev/msr268.

5 Siegfried Dubois, Matthias Zytnicki, Claire Lemaitre, and Thomas Faraut. Pairwise graph edit distance characterizes the impact of the construction method on pangenome graphs. *Bioinformatics*, 41(6), May 2025. doi:10.1093/bioinformatics/btaf291.

6 Erik Garrison, Andrea Guarracino, Simon Heumos, Flavia Villani, Zhigui Bao, Lorenzo Tattini, Jörg Hagmann, Sebastian Vorbrugg, Santiago Marco-Sola, Christian Kubica, David G. Ashbrook, Kaisa Thorell, Rachel L. Rusholme-Pilcher, Gianni Liti, Emilio Rudbeck, Agnieszka A. Golicz, Sven Nahnsen, Zuyu Yang, Moses Njagi Mwaniki, Franklin L. Nobrega, Yi Wu, Hao Chen, Joep de Ligt, Peter H. Sudmant, Sanwen Huang, Detlef Weigel, Nicole Soranzo, Vincenza Colonna, Robert W. Williams, and Pjotr Prins. Building pangenome graphs. *Nature Methods*, 21(11):2008–2012, October 2024. doi:10.1038/s41592-024-02430-3.

7 Erik Garrison, Jouni Sirén, Adam M Novak, Glenn Hickey, Jordan M Eizenga, Eric T Dawson, William Jones, Shilpa Garg, Charles Markello, Michael F Lin, Benedict Paten, and Richard Durbin. Variation graph toolkit improves read mapping by representing genetic variation in the reference. *Nat Biotechnol*, 36(9):875–879, October 2018. doi:10.1038/nbt.4227.

**8** Andrea Guarracino. Gfalace, July 2025. URL: `https://github.com/pangenome/gfalace`.

**9** Glenn Hickey, Jean Monlong, Jana Ebler, Adam M. Novak, Jordan M. Eizenga, Yan Gao, Haley J. Abel, Lucinda L. Antonacci-Fulton, Mobin Asri, Gunjan Baid, Carl A. Baker, Anastasiya Belyaeva, Konstantinos Billis, Guillaume Bourque, Silvia Buonaiuto, Andrew Carroll, Mark J. P. Chaisson, Pi-Chuan Chang, Xian H. Chang, Haoyu Cheng, Justin Chu, Sarah Cody, Vincenza Colonna, Daniel E. Cook, Robert M. Cook-Deegan, Omar E. Cornejo, Mark Diekhans, Daniel Doerr, Peter Ebert, Jana Ebler, Evan E. Eichler, Susan Fairley, Olivier Fedrigo, Adam L. Felsenfeld, Xiaowen Feng, Christian Fischer, Paul Flicek, Giulio Formenti, Adam Frankish, Robert S. Fulton, Shilpa Garg, Erik Garrison, Nanibaa' A. Garrison, Carlos Garcia Giron, Richard E. Green, Cristian Groza, Andrea Guarracino, Leanne Haggerty, Ira M. Hall, William T. Harvey, Marina Haukness, David Haussler, Simon Heumos, Kendra Hoekzema, Thibaut Hourlier, Kerstin Howe, Miten Jain, Erich D. Jarvis, Hanlee P. Ji, Eimear E. Kenny, Barbara A. Koenig, Alexey Kolesnikov, Jan O. Korbel, Jennifer Kordosky, Sergey Koren, HoJoon Lee, Alexandra P. Lewis, Wen-Wei Liao, Shuangjia Lu, Tsung-Yu Lu, Julian K. Lucas, Hugo Magalhães, Santiago Marco-Sola, Pierre Marijon, Charles Markello, Tobias Marschall, Fergal J. Martin, Ann McCartney, Jennifer McDaniel, Karen H. Miga, Matthew W. Mitchell, Jacquelyn Mountcastle, Katherine M. Munson, Moses Njagi Mwaniki, Maria Nattestad, Sergey Nurk, Hugh E. Olsen, Nathan D. Olson, Trevor Pesout, Adam M. Phillippy, Alice B. Popejoy, David Porubsky, Pjotr Prins, Daniela Puiu, Mikko Rautiainen, Allison A. Regier, Arang Rhie, Samuel Sacco, Ashley D. Sanders, Valerie A. Schneider, Baergen I. Schultz, Kishwar Shafin, Jonas A. Sibbesen, Jouni Sirén, Michael W. Smith, Heidi J. Sofia, Ahmad N. Abou Tayoun, Françoise Thibaud-Nissen, Chad Tomlinson, Francesca Floriana Tricomi, Flavia Villani, Mitchell R. Vollger, Justin Wagner, Brian Walenz, Ting Wang, Jonathan M. D. Wood, Aleksey V. Zimin, Justin M. Zook, Tobias Marschall, Heng Li, and Benedict Paten. Pangenome graph construction from genome alignments with minigraph-cactus. *Nature Biotechnology*, 42(4):663–673, May 2023. `doi:10.1038/s41587-023-01793-w`.

**10** Ilia Minkin and Paul Medvedev. Scalable multiple whole-genome alignment and locally collinear block construction with SibeliaZ. *Nat Commun*, 11(1):6327, December 2020. `doi:10.1038/s41467-020-19777-8`.

**11** Adam M Novak, Erik Garrison, and Benedict Paten. A graph extension of the positional burrows-wheeler transform and its applications. *Algorithms Mol Biol*, 12:18, July 2017. `doi:10.1186/s13015-017-0109-9`.

**12** Mikko Rautiainen and Tobias Marschall. GraphAligner: rapid and versatile sequence-to-graph alignment. *Genome Biol*, 21(1):253, September 2020. `doi:10.1186/s13059-020-02157-2`.

**13** Jouni Sirén. Indexing variation graphs. In Sándor Fekete and Vijaya Ramachandran, editors, *2017 Proceedings of the Ninteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 13–27, Philadelphia, PA, January 2017. Society for Industrial and Applied Mathematics. `doi:10.1137/1.9781611974768.2`.

# PRISM: Partition-Function Decomposition into Structural Classes for Hierarchically Constrained RNA Pseudoknot Ensembles

**Mateo Gray**
Department of Biomedical Engineering, University of Alberta, Edmonton, Canada

**Sebastian Will**
Department of Computer Science, Institut Polytechnique de Paris, Palaiseau, France

**Hosna Jabbari**[1]
Department of Biomedical Engineering, University of Alberta, Edmonton, Canada

## Abstract

While structure ensemble analysis became a valuable routinely applied tool for pseudoknot-free RNA, the extension to pseudoknots remains challenging due to the computational hardness of the general problem. The existing efficient algorithms for the computation of partition function with pseudoknots were still computationally expensive and were restricted to simple pseudoknots. This changed only with CParty, which computes pseudoknotted partition functions with the efficiency of pseudoknot-free folding. At its core, CParty follows the hierarchical folding hypothesis, such that ensemble structures can form pseudoknots only with a given input constraint structure. For an RNA sequence $S$ and pseudoknot-free structure $G$, CParty limits the ensemble to "density-2" structures $G \cup G'$ for a second, disjoint pseudoknot-free structure $G'$.

We present PRISM that extends CParty from pure partition function calculation to full-fledged posterior probability analysis. By stochastic traceback through CParty's dynamic programming matrices, it samples structures from the conditional Boltzmann ensemble. From estimated base pair probabilities, it generates ensemble representations, predicts centroid and maximum expected accuracy structures and calculates properties. In addition to position-specific summaries, PRISM maps sampled structures to RNA shapes, producing a posterior distribution over topological abstractions. This shape-level summary captures ensemble diversity even when a conserved pseudoknotted motif appears with shifted base-pair positions across samples.

We validate PRISM in the pseudoknot-free limit, where it reproduces RNAFold quantities for minimum free energy, ensemble free energy, centroid expected distance, and maximum expected accuracy. We further show that stochastic traceback recovers Boltzmann structure probabilities and that sampling error decreases at the expected Monte Carlo rate while runtime grows linearly with the number of samples. Our case study demonstrate that RNA-shape summaries can reveal dominant pseudoknotted topologies that centroid decoding may miss. PRISM thus converts the CParty partition function into a practical framework for posterior decoding and topology-aware analysis of hierarchically constrained pseudoknotted RNA ensembles.

**2012 ACM Subject Classification** Applied computing → Computational biology

**Keywords and phrases** RNA, MFE, Secondary Structure Prediction, Pseudoknot, Partition Function, Centroid, MEA, RNA shape, Stochastic traceback

**Digital Object Identifier** 10.4230/LIPIcs.WABI.2026.30

**Supplementary Material** *Software (PRISM's implementation and detailed Results)*: `https://github.com/TheCOBRALab/PRISM` [10]
archived at `swh:1:dir:79803e70b92d8457ba08d16fe67b09382501716c`
*Dataset*: `https://github.com/mateog4712/PRISM-RawData` [11]
archived at `swh:1:dir:4646c334d25f499a6485f62fe913bb90e2cdd115`

[1] Corresponding author

© Mateo Gray, Sebastian Will, and Hosna Jabbari;
licensed under Creative Commons License CC-BY 4.0

## 1 Introduction

RNA molecules perform diverse biological functions whose mechanisms often depend on their structural conformations [5, 17, 21, 30, 32]. Structured RNA regions can regulate translation, control pre-mRNA splicing, mediate catalytic activity, and organize long-range transcript architectures. In these settings, RNA structure is not only a consequence of sequence composition, but part of the molecular mechanism by which an RNA exposes or hides functional elements, recognizes binding partners, stabilizes alternative conformations, or catalyzes chemical reactions. Computational models of RNA structure are therefore central to the interpretation of RNA function, the prioritization of experimental hypotheses, and the identification of structural elements relevant to disease or therapeutic design. Because experimental structure determination remains labor-intensive, condition-dependent, and difficult to scale to transcriptome-wide settings, computational prediction and ensemble analysis remain indispensable complements to experimental approaches.

**RNA structure ensembles and pseudoknot classes.** Let $S = s_1 \cdots s_n$ be an *RNA sequence* of $n$ *nucleotides* (also called, *bases*). An *RNA secondary structure* $R$ is a set of *base pairs* $(i, j)$, $1 \le i < j \le n$, such that each nucleotide participates in at most one pair. Two base pairs $(i, j)$ and $(k, \ell)$ *cross* if $i < k < j < \ell$ or $k < i < \ell < j$. A structure is considered *pseudoknot-free* if it contains no crossing base pair; otherwise, it is *pseudoknotted*. For a secondary structure class $\mathcal{C}(S)$, secondary structure prediction algorithms assign a *free energy* $E_S(R)$ to each $R \in \mathcal{C}(S)$. For a structure $R$ of $S$, define its *Boltzmann weight* $w_S(R) = \exp(-E_S(R)/(\mathcal{R}T))$, depending on the absolute temperature $T$ via the gas constant $\mathcal{R}$. Then, the *partition function* $Z_{\mathcal{C}}(S) = \sum_{R \in \mathcal{C}(S)} w_S(R)$ defines the *Boltzmann distribution* in the *Boltzmann ensemble* $\mathcal{C}(S)$ by

$$\mathbf{P}_{\mathcal{C}}[R \mid S] = \frac{w_S(R)}{Z_{\mathcal{C}}(S)} \qquad \text{for } R \in \mathcal{C}(S).$$

Note that the structures with *minimum free energy (mfe)* $\min_{R \in \mathcal{C}} E_S(R)$ have maximum probability in this distribution. The *posterior probability* of the base pair $(i, j)$ is defined as

$$\mathbf{P}_{\mathcal{C}}\big[(i,j) \in R\big] = \frac{1}{Z_{\mathcal{C}}(S)} \sum_{\substack{R \in \mathcal{C}(S) \\ (i,j) \in R}} w_S(R).$$

Such base pair probabilities characterize the base pairing propensities in the thermodynamic equilibrium beyond reporting single predicted structures and are oftentimes used for visualization in dot plots.

**Ensemble analysis.** For the class of pseudoknot-free structures, established dynamic-programming algorithms provide the toolkit for computing partition functions, base-pair probabilities, stochastic traceback samples, centroid structures, and maximum expected accuracy (MEA) structure [20, 7, 6, 8, 18]. These posterior summaries are often more informative than a single MFE structure as they distinguish well-supported base pairs from uncertain or competing alternatives.

Pseudoknots substantially complicate this ensemble analysis problem. Predicting the minimum free energy secondary structure of pseudoknotted RNA is NP-hard in general [19, 1] and even inapproximable [25]. Existing pseudoknotted MFE prediction algorithms, therefore restrict the class of structures they handle or simplify the energy model [16, 13, 14].

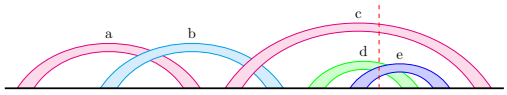



**Figure 1** Bands of a density-2 structure. A pseudoknot can be decomposed into bands, each of which is internally non-crossing but crosses the remaining structure. All base pairs within a band share the same crossing pattern with respect to the rest of the structure. Bands that mutually cross one another are grouped into components, here {a,b,c} and {d,e}. A set of bands is density-$k$, if every position is covered by at most $k$ bands of the same component. An example of this is shown at the dashed red line [15, 16].

**Hierarchical folding: HFold and CParty.** Particularly fast pseudoknot folding algorithms were proposed based on the biologically motivated hierarchical folding hypothesis: an RNA molecule first folds into a pseudoknot-free structure; additional base pairs are subsequently added to further minimize the structure's energy, allowing the resulting structure to contain crossing base pairs [28]. The tool HFold for hierarchical pseudoknot prediction, finds the MFE structure given a pseudoknot-free input structure $G$ [13, 14]. HFold's class of structures is called *density-2* structures (Fig. 1), a subclass of general bisecondary structures [33]. This large structure class includes kissing hairpins and chains of pseudoknots; moreover, it recursively allows density-2 substructures.

CParty extended HFold's hierarchical model from MFE prediction to partition-function computation [9]. Given a sequence $S$ and a fixed pseudoknot-free input structure $G$, CParty considers all structures of the form $G \cup G'$, where $G'$ is pseudoknot-free, $G \cap G' = \emptyset$, and $G \cup G'$ is a valid density-2 structure. Equivalently, CParty defines the conditional ensemble

$$\Omega[S, G] = \{G \cup G' : G' \text{ is pseudoknot-free}, \ G \cap G' = \emptyset, \ G \cup G' \text{ is density-2}\},$$

and computes $Z(S \mid G) = \sum_{R \in \Omega[S,G]} w_S(R)$. The induced conditional Boltzmann distribution is

$$\mathbf{P}_{\Omega[G]}[R \mid S, G] = \frac{w_S(R)}{Z(S \mid G)} \qquad \text{for } R \in \Omega[S, G].$$

Thus, CParty provides the normalizing constant and constrained ensemble free energy for $\Omega[S, G]$. However, the scalar value $Z(S \mid G)$ does not by itself identify which base pairs are well supported, which nucleotides are likely unpaired, whether the posterior mass is concentrated on pseudoknotted or pseudoknot-free extensions, or whether distinct base-pair-level structures represent the same higher-level topology.

**Pseudoknot ensemble analysis: PRISM.** In this work, we introduce PRISM, a posterior-decoding and ensemble-analysis framework built on CParty. The name PRISM reflects the central operation of the method: whereas CParty computes one constrained partition function, PRISM separates the same Boltzmann ensemble into interpretable structural classes and estimates the posterior mass of those classes. Here, a structural class is any subset $\mathcal{A} \subseteq \Omega[S, G]$ defined by a structural property of interest. For example, $\mathcal{A}$ may be the set of structures containing a particular base pair, the set of structures in which a nucleotide is unpaired, the set of structures containing at least one pseudoknot, or the set of structures with a particular RNA shape. The Boltzmann weight assigned to such a class is $Z_{\mathcal{A}}(S \mid G) = \sum_{R \in \mathcal{A}} w_S(R)$, and its posterior probability under the conditional ensemble is

$$\mathbf{P}_{\Omega[G]}[R \in \mathcal{A} \mid S, G] = \frac{Z_{\mathcal{A}}(S \mid G)}{Z(S \mid G)}.$$

If a collection of classes $\mathcal{A}_1, \ldots, \mathcal{A}_m$ is disjoint and covers $\Omega[S, G]$, then the CParty partition function decomposes as

$$Z(S \mid G) = \sum_{\ell=1}^{m} Z_{\mathcal{A}_\ell}(S \mid G).$$

Thus, the same partition function can be viewed not only as a scalar normalizing constant, but also as the total Boltzmann weight distributed across structural classes.

PRISM does not recompute these restricted partition functions exactly. Instead, it draws stochastic traceback samples from $\mathbf{P}_G(\cdot \mid S, G)$ and estimates class posterior probabilities by sample frequencies. For a structural class $\mathcal{A}$, the Monte Carlo estimate is

$$\widehat{\mathbf{P}}[\mathcal{A} \mid S, G] = \frac{1}{N} \sum_{t=1}^{N} \mathbf{1}\left\{R^{(t)} \in \mathcal{A}\right\}.$$

The classes considered in this paper include base-pair events, unpaired-nucleotide events, pseudoknot-conditioned subensembles, decoded structures, and RNA-shape abstractions. PRISM estimates base-pair and unpaired probabilities, constructs a marginal structural-state string, computes centroid and pseudoknot-conditioned centroid structures, computes the MEA structure, and summarizes the ensemble at the level of RNA shapes [26]. The shape-level view is especially useful when a pseudoknotted motif is conserved topologically but its exact base-pair endpoints shift across sampled structures.

**Contributions.** We introduce PRISM, an algorithmic posterior analysis layer for the hierarchically constrained density-2 ensemble computed by CParty. We first formalize posterior analysis for this ensemble by making explicit the conditional Boltzmann distribution over admissible extensions $G \cup G'$. We then introduce a posterior class-weight view of the CParty partition function: rather than using the partition function only as a scalar normalizing constant, PRISM estimates how its Boltzmann weight is distributed across interpretable structural classes, including base-pair events, unpaired-nucleotide events, pseudoknot-conditioned subensembles, decoded structures, and RNA-shape abstractions. From these probabilities, PRISM computes posterior summaries for the constrained pseudoknot ensemble, including a marginal structural-state string, centroid structures, pseudoknot-conditioned centroid structures, maximum expected accuracy structures, and dot-plot visualizations. Finally, we introduce shape-level summaries of sampled structures, which provide a topology-aware view of ensemble diversity that is robust to small shifts in helix position. We validate the implementation in the pseudoknot-free limit against RNAFold, examine sampling convergence and runtime overhead, and show that RNA-shape summaries can reveal dominant pseudoknotted ensemble features that base-pair-level centroid decoding may miss.

## 2 Methods

The input to PRISM is an RNA sequence $S = s_1 \cdots s_n$ and a fixed pseudoknot-free input structure $G$. PRISM utilizes the conditional partition function $Z(S \mid G)$ over the admissible density-2 ensemble $\Omega[S, G]$, and samples structures $R^{(1)}, R^{(2)}, \ldots, R^{(N)} \in \Omega[S, G]$ from the conditional Boltzmann distribution $\mathbf{P}_{\Omega[G]}(R \mid S, G) = \frac{w_S(R)}{Z(S|G)}$. Unless otherwise stated, we use $N = 1000$ stochastic traceback samples to estimate ensemble statistical properties.

Let $\mathcal{B}_G(S)$ denote the set of base pairs that can occur in at least one structure in $\Omega[S, G]$. The posterior probability of a base pair $(i, j) \in \mathcal{B}_G(S)$ in the conditional ensemble is defined in analogy to unconditional base pair probabilities as

$$p_{ij} = \mathbf{P}_{\Omega[G]}[(i, j) \mid S, G] = \sum_{R \in \mathcal{C}(S), (i,j) \in R} \mathbf{P}_{\Omega[G]}[R \mid S, G].$$

Although the probabilities $p_{ij}$ conditionally depend on $G$, we abstain from more verbose notation like $p_{i,j|G}$, since $G$ will be always clear from the context. Note that in practice, PRISM always estimates base pair probabilities from sampling by stochastic traceback and operates on estimates $\widehat{p}_{ij}$. In the following, we use the hat notation only where this distinction is directly relevant.

## 2.1 Stochastic traceback

For pseudoknot-free RNA structures, exact base-pair probabilities can be computed by dynamic-programming recurrences derived from the partition function [20, 18]. An alternative is representative sampling: structures are drawn from the Boltzmann ensemble, and base-pair frequencies in the sample are used as Monte Carlo estimates of posterior base-pair probabilities [7]. PRISM extends this representative-sampling strategy to the hierarchically constrained ensemble computed by CParty.

Stochastic traceback follows the same decomposition structure as a standard dynamic-programming traceback. At each traceback step, however, the algorithm samples one of the possible recurrence cases according to its conditional Boltzmann weight. Suppose a dynamic-programming state $A$ has recurrence $Z_A = \sum_{c \in \mathcal{D}(A)} W_c$, where $\mathcal{D}(A)$ is the set of decomposition cases for $A$, and $W_c$ is the Boltzmann contribution of case $c$, including the weights of any subproblems introduced by that case. During traceback from state $A$, PRISM chooses case $c$ with probability

$$\mathbf{P}[c \mid A] = \frac{W_c}{Z_A}.$$

The algorithm then recursively traces back through the subproblems specified by the selected case. At the top level, this procedure samples a complete structure $R \in \Omega[S, G]$ with probability $w_S(R)/Z(S \mid G)$. Thus, the stochastic traceback samples are drawn from the same conditional Boltzmann distribution whose partition function is computed by CParty.

The correctness of this sampling rule follows from the usual recursive decomposition argument. Each dynamic-programming state is a Boltzmann-weighted sum over mutually exclusive decomposition cases. Choosing each case with probability equal to its contribution divided by the state partition function makes the probability of any complete traceback path equal to the product of the conditional probabilities along that path. The intermediate state partition functions cancel telescopically, leaving exactly the Boltzmann weight of the completed structure divided by $Z(S \mid G)$.

Operationally, PRISM draws a uniform random number and compares it with the cumulative Boltzmann weights of the recurrence cases. To reduce the cost of repeated cumulative-weight scans, we use the Boustrophedon sampling strategy [22]. This improves the efficiency of sampling without changing the target Boltzmann distribution.

Given $N$ sampled structures, the posterior probability of base pair $(i, j)$ is estimated as

$$\widehat{p}_{ij} = \frac{f_{ij}}{N} = \frac{1}{N} \sum_{t=1}^{N} \mathbf{1}\left\{(i, j) \in R^{(t)}\right\},$$

where $f_{ij}$ is the number of sampled structures containing $(i, j)$. This directly yields estimates of the unpaired probabilities $p_i^u$, since

$$p_i^u = 1 - \sum_{j<i} p_{ji} - \sum_{j>i} p_{ij}.$$

## 2.2 Marginal structural-state string

PRISM constructs a marginal structural-state string from the estimated posterior probabilities. This string is a position-wise annotation of the ensemble: for each nucleotide, PRISM reports the most probable local structural state of that nucleotide. It is not a globally decoded RNA structure, and its brackets are not required to define a valid set of base pairs.

We use the alphabet { `.` `(`, `)`, `[`, `]` } corresponding respectively to the five local states: unpaired (u), non-crossing left endpoint (NL) non-crossing right endpoint (NR), crossing left endpoint (CL), and crossing right endpoint (CR). The classification of a base pair as crossing or non-crossing is understood relative to the fixed input structure $G$. Let

$$\chi_G(i,j) = \begin{cases} 1, & \text{if } (i,j) \text{ crosses at least one base pair in } G, \\ 0, & \text{otherwise.} \end{cases}$$

All base pairs in $G$ are therefore classified as non-crossing, since $G$ itself is pseudoknot-free. For each position $i$, PRISM computes

$$\widehat{p}_i^{\,\mathrm{NL}} = \sum_{\substack{j>i \\ \chi_G(i,j)=0}} \widehat{p}_{ij}, \qquad \widehat{p}_i^{\,\mathrm{NR}} = \sum_{\substack{j<i \\ \chi_G(j,i)=0}} \widehat{p}_{ji},$$

and

$$\widehat{p}_i^{\,\mathrm{CL}} = \sum_{\substack{j>i \\ \chi_G(i,j)=1}} \widehat{p}_{ij}, \qquad \widehat{p}_i^{\,\mathrm{CR}} = \sum_{\substack{j<i \\ \chi_G(j,i)=1}} \widehat{p}_{ji}.$$

The symbol at position $i$ is chosen by taking the maximum among

$$\widehat{p}_i^{\,u}, \quad \widehat{p}_i^{\,\mathrm{NL}}, \quad \widehat{p}_i^{\,\mathrm{NR}}, \quad \widehat{p}_i^{\,\mathrm{CL}}, \quad \widehat{p}_i^{\,\mathrm{CR}}.$$

Thus, the marginal structural-state string summarizes the most likely state of each nucleotide under the conditional ensemble. Unlike a sampled structure, centroid structure, or MEA structure, it is not obtained by optimizing a global structural objective; it reports local marginal posterior states.

## 2.3 Ensemble diversity

PRISM computes ensemble diversity from the estimated base-pair probabilities. In analogy to the common definition for non-crossing ensembles, we define the ensemble diversity $\langle d \rangle$ of the conditional ensemble as expected distance between two structures

$$\langle d \rangle(S \mid G) = \sum_{R \in \Omega[G], R' \in \Omega[G]} \mathbf{P}[R \mid S, G] \cdot \mathbf{P}[R' \mid S, G] \cdot d_{\mathrm{BP}}(R, R')$$

Defining the base pair distance $d_{\mathrm{BP}}$ as symmetric difference

$$d_{\mathrm{BP}}(R, R') = |R \setminus R'| + |R' \setminus R|,$$

let us compute ensemble diversity from base pair probabilities

$$\langle d \rangle(S \mid G) = \sum_{(i,j) \in \mathcal{B}_G(S)} 2 \cdot p_{ij} \left(1 - p_{ij}\right).$$

This statistic is small when most admissible base pairs have posterior probability near 0 or 1, and it increases when posterior mass is spread across competing alternatives. Each term is maximized at $p_{ij} = 1/2$, so the statistic emphasizes uncertain or mutually competing contacts.

## 2.4 Centroid structures

The centroid structure is a posterior decoder that minimizes expected base-pair distance to the ensemble [6]. Using posterior base-pair probabilities, the expected distance from $Y \in \Omega[S, G]$ to the conditional ensemble is

$$\widehat{d}_G(Y) = \sum_{R \in \Omega[S,G]} \mathbf{P}_{\mathcal{C}}[R \mid S, G] \cdot d_{\mathrm{BP}}(R, Y) = \sum_{(i,j) \in Y} (1 - p_{ij}) + \sum_{(i,j) \in \mathcal{B}_G(S) \setminus Y} p_{ij}.$$

Thus, we obtain the centroid structure $Y^*$ that minimizes this expected distance by selecting all base pairs $(i, j)$ with $p_{ij} > 0.5$ and excluding all others.

Note that the analogous construction of the centroid in the pseudoknot-free case, necessarily yields a pseudoknot-free centroid structure. In the case of PRISM, where $p_{i,j}$ are probabilities in the conditional ensemble of $G$, one observes the following properties: Since $G$ is fixed, base pairs in $G$ have probability 1. For the remaining base pairs of the centroid $Y^*$ in $Y^* \setminus G$, we can argue as in the non-crossing case, that they cannot cross each other. Therefore, the centroid structure is bi-secondary. Note that we forgo the stricter claim for density-2 in this contribution to avoid technical digressions.

**Pseudoknot-conditioned centroid.** PRISM also computes a pseudoknot-conditioned centroid. This decoder uses the same centroid objective, but estimates probabilities only from sampled structures that contain at least one crossing pair. Let

$$\mathcal{T}_{\mathrm{PK}} = \left\{ t \in \{1, \ldots, N\} : R^{(t)} \text{ contains at least one crossing pair} \right\},$$

and let $N_{\mathrm{PK}} = |\mathcal{T}_{\mathrm{PK}}|$. If $N_{\mathrm{PK}} = 0$, no pseudoknot-conditioned centroid is reported. Otherwise, the pseudoknot-conditioned base-pair probability is estimated as

$$\widehat{p}_{ij}^{\,\mathrm{PK}} = \frac{1}{N_{\mathrm{PK}}} \sum_{t \in \mathcal{T}_{\mathrm{PK}}} \mathbf{1}\left\{ (i, j) \in R^{(t)} \right\}.$$

The pseudoknot-conditioned centroid is computed by applying the same expected-distance decoder to $\widehat{p}_{ij}^{\,\mathrm{PK}}$ instead of $\widehat{p}_{ij}$. This conditioning removes sampled structures without pseudoknots from the probability estimates, so pseudoknotted alternatives are not diluted by pseudoknot-free structures.

This decoder still operates at the level of exact base-pair identities. If a pseudoknotted helix is present in many sampled structures but shifts by one or two nucleotides across samples, its probability can be split among nearby contacts such as $(i, j)$, $(i, j-1)$, and $(i+1, j)$. In that case, no individual crossing base pair may exceed the 50% centroid threshold, even though the pseudoknotted motif is common in the ensemble. This limitation motivates the shape-level summaries described below.

## 2.5 Maximum expected accuracy structure

The maximum expected accuracy (MEA) decoder selects a structure that maximizes the expected number of correctly predicted paired and unpaired positions [8]. For a candidate structure $Y$, let $U(Y)$ be the set of nucleotides unpaired in $Y$. Given a base-pair weight parameter $\gamma > 0$, PRISM defines the expected accuracy of $Y$ as

$$EA_\gamma(Y) = \sum_{(i,j) \in Y} 2\gamma p_{ij} + \sum_{i \in U(Y)} p_i^u.$$

The MEA structure is $Y_{\mathrm{MEA}} = \arg\max_Y EA_\gamma(Y)$, where the maximization is over structures compatible with the hierarchical decomposition.

PRISM uses the $G \cup G'$ form of hierarchical folding to reduce this calculation to the standard pseudoknot-free MEA calculation. Since $G$ is fixed, its base pairs are present in every admissible candidate structure. The variable part of the prediction is the pseudoknot-free extension $G'$. PRISM therefore optimizes the MEA objective over the pseudoknot-free extension and then adds the fixed input structure $G$. Base pairs in $G$ contribute a constant term to the objective, and nucleotides paired in $G$ are not eligible to be unpaired in the candidate structure. The pseudoknot-free MEA recurrence is standard [8, 18], so we do not reproduce it here.

## 2.6 RNA shape abstraction

RNA shapes provide an abstraction of secondary structure that groups structures according to higher-level topology rather than exact base-pair positions [26, 23]. This is useful for posterior analysis because small shifts in helix endpoints can produce many distinct base-pair-level structures while preserving the same overall structural organization.

For pseudoknot-free structures, Steffen et al. [26] defined several abstraction levels. At low abstraction levels, some unpaired regions and loop details are retained. At abstraction level 5, unpaired nucleotides are removed and stacks and internal loops are collapsed so that only the nesting pattern of helices remains. For example,

```
((.((..((...)))).((...))..)).
```

(Level 1) `(_(_(_))(_)_)_`

(Level 5) `(()())`

PRISM uses an analogous level-5 abstraction for pseudoknotted structures. Given a sampled structure $R$, we represent non-crossing base pairs using `()` and base pairs that cross the input structure $G$ using `[]`. We then delete unpaired nucleotides and collapse each maximal stack of base pairs into a single base pair, preserving the order and crossing pattern of the remaining helices. Both crossing and non-crossing helices are retained. We denote the resulting shape by $\sigma_5(R)$.

This construction is related to, but distinct from, the *shadow* abstraction of Reidys et al. [23]. Their shadow construction removes non-crossing arcs, removes isolated vertices, and collapses the remaining stacks. In our notation, this corresponds to a further abstraction, denoted $\sigma_6(R)$, in which non-crossing helices are removed after the level-5 representation. For example,

```
((..((...))..[[..)).]]
```

(Level 5) `(()[])`

(Level 6) `([])`

Thus, $\sigma_5$ retains both the pseudoknotted and non-pseudoknotted components of the topology, whereas $\sigma_6$ focuses only on the crossing core. We use $\sigma_5$ as the default shape abstraction for ensemble summaries because it preserves more information about the complete structural organization while still removing local positional variation.

Given sampled structures $R^{(1)}, \ldots, R^{(N)}$, PRISM estimates the posterior probability of each shape $a$ by

$$\widehat{q}(a) = \frac{1}{N} \sum_{t=1}^{N} \mathbf{1}\Big\{\sigma_5(R^{(t)}) = a\Big\}.$$

The resulting distribution over shapes characterizes the topology-level diversity of the conditional ensemble. Structures that differ by small shifts in base-pair endpoints can map to the same shape, so shape frequencies can reveal dominant pseudoknotted motifs even when no individual shifted base pair has sufficiently high probability to appear in the centroid.

## 3 Datasets and input constraints

For the empirical time and space analysis of PRISM, we obtained 2824 sequences from the RNASTRAND V2.0 database [2]. The median sequence length was 242 nucleotides, and the longest sequence contained 1500 nucleotides. For each sequence, we identified the 20 most stable stem-loop structures by scoring candidate stem loops using hairpin and stacking base-pair energies across the sequence. These stem-loop structures were used as fixed input constraints $G$ for PRISM.

For the shape-based case study, we used the ZMP–ZTP riboswitch family from Rfam, identified by RF01750 [12]. We partitioned the Rfam consensus structure for this family into its pseudoknot-free and pseudoknotted components. We denote the pseudoknot-free component by $G_{\text{big}}$ and the pseudoknotted component by $G_{\text{small}}$. Each of these two consensus-derived structures was fitted to each ungapped sequence in the family, and the fitted structures were used as alternative input constraints for PRISM.

## 4 Empirical Results

We evaluate PRISM in three stages. First, we validate correctness in the pseudoknot-free limit $G = \emptyset$, where $\Omega[S, G]$ is the standard secondary-structure ensemble and PRISM should agree with RNAFold under the same energy model. This validation includes both deterministic posterior summaries and stochastic traceback: we compare MFE energy, ensemble free energy, centroid expected distance, and MEA score, and we verify that the observed frequency of the MFE structure in sampled structures matches its Boltzmann probability. Second, we measure empirical runtime and memory usage for the full PRISM workflow and quantify how the stochastic traceback sample count affects runtime and posterior-estimation error. Third, we evaluate topology-level summaries by mapping sampled structures to level-5 RNA shapes. These shape summaries demonstrate how PRISM separates the conditional partition function into interpretable structural classes and can reveal pseudoknotted motifs or alternative conformations that are not apparent from base-pair-level centroid decoding alone.

### 4.1 PRISM agrees with RNAFold in the pseudoknot-free limit

When the input structure is empty, $G = \emptyset$, the conditional ensemble $\Omega[S, G]$ reduces to the standard pseudoknot-free secondary-structure ensemble. Consequently, $Z(S \mid \emptyset)$ should agree with the pseudoknot-free partition function computed by RNAFold under the same energy model and parameter settings. As a first validation of the implementation, we compared the MFE energy, ensemble free energy, centroid distance, and MEA score computed by PRISM with the corresponding quantities computed by RNAFold [18]. All comparisons used DP09 parameters [3] without dangle energies. PRISM reproduced the RNAFold values to numerical precision across all four quantities (Fig. 2).

We next validated the stochastic traceback procedure. For a structure $R$, the Boltzmann probability predicted by the partition function is $w_S(R)/Z(S \mid G)$. Therefore, for the MFE structure $R_{\text{MFE}}$, the observed frequency of $R_{\text{MFE}}$ among stochastic traceback samples should match $\frac{w_S(R_{\text{MFE}})}{Z(S|G)}$. We performed this comparison for sequences with length $n \leq 100$. Longer

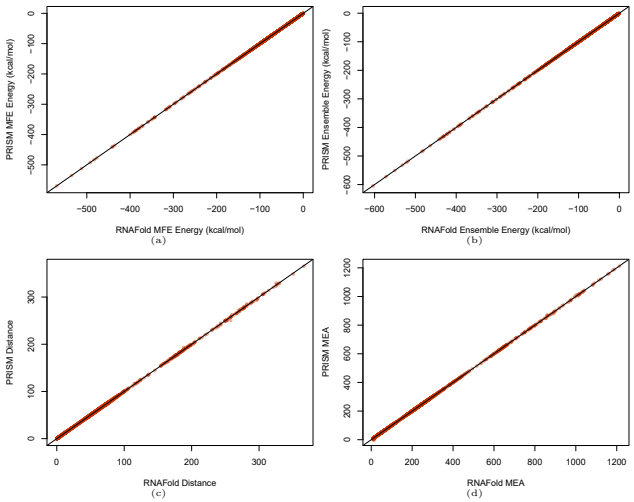



**Figure 2** Validation of PRISM against RNAFold in the pseudoknot-free limit $G = \emptyset$. For each sequence, the PRISM output is plotted against the RNAFold output under the same energy parameters. Panels show (a) MFE energy, (b) ensemble free energy, (c) centroid expected distance to the ensemble, and (d) MEA score. Points on the diagonal indicate agreement between the two implementations.

sequences were excluded from this validation because the Boltzmann mass of any single structure, including the MFE structure, is typically very small, making direct frequency comparisons less informative at the sample sizes used. The observed MFE frequencies closely matched their partition-function probabilities (Fig. 3), supporting that stochastic traceback samples from the intended conditional Boltzmann distribution.

## 4.2 PRISM has empirical time and memory scaling close to RNAFold

We next evaluated the empirical time and memory requirements of the PRISM workflow when calculating pseudoknot partition functions. The measured workflow included computation of the constrained partition function $Z(S \mid G)$, stochastic traceback, centroid decoding, and MEA decoding. We compared PRISM's performance with the pseudoknot-free RNAFold and with NUPACK 3.0. To our knowledge, NUPACK is the only available a tool for computing unconstrained pseudoknot partition functions. It considers the class of simple pseudoknots in $\mathrm{O}(n^5)$ time and $\mathrm{O}(n^4)$ space complexity [34]. Since PRISM requires both a sequence $S$ and an input structure $G$, we used the stem-loop constraints generated as described in Section 3. The same constrained inputs were provided to RNAFold; NUPACK 3.0 does not support structure constraints and was run on sequence alone.

All experiments were run on an M2 Max Mac Studio with 64 GB of RAM. Runtime was measured as user CPU time, and memory usage was measured as maximum resident set size. Across the benchmark, the maximum runtime observed for PRISM was 23.608 seconds, and the maximum memory usage was 293024 KB. For RNAFold, the corresponding maximum runtime and memory usage were 7.674 seconds and 56256 KB, respectively.

The empirical scaling behavior is shown in Fig. 4. For sequence lengths $n \geq 250$, the fitted runtime exponent for PRISM was close to that of RNAFold and substantially below that of NUPACK 3.0 over the shared range of lengths. The fitted memory exponent showed the same qualitative pattern. Thus, despite extending the pseudoknot-free posterior-analysis toolkit to the hierarchically constrained density-2 ensemble, PRISM remains close to RNAFold in empirical scaling and below NUPACK 3.0 in both runtime and memory usage on this benchmark.

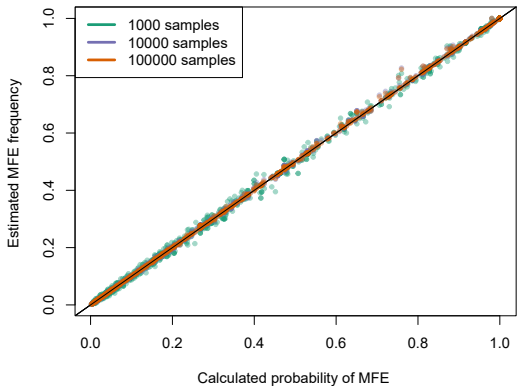



**Figure 3** Validation of stochastic traceback using the MFE structure. The x-axis gives the Boltzmann probability of the MFE structure computed from its contribution to the partition function, and the y-axis gives the observed frequency of the MFE structure in stochastic traceback samples. Results are shown for sequences with $n \leq 100$. The diagonal line $y = x$ denotes exact agreement.

## 4.3 Sampling introduces linear runtime overhead and Monte Carlo convergence

Because PRISM estimates base-pair probabilities from stochastic traceback samples, the number of samples controls a tradeoff between runtime and posterior-estimation accuracy. To quantify this tradeoff, we evaluated three representative sequence lengths: 100, 210, and 320. For each length, we measured runtime as a function of the number of samples $N_{\text{samp}}$. Runtime increased linearly with $N_{\text{samp}}$, consistent with a fixed cost for filling the dynamic-programming tables followed by a per-sample traceback cost (Fig. 5a).

We also measured the accuracy of estimated base-pair probabilities as a function of $N_{\text{samp}}$. For each sample count, we computed the root mean square deviation (RMSD) between the base-pair probabilities estimated from that number of samples and the corresponding estimates obtained from 100000 samples, which were used as a reference:

$$\text{RMSD}(N_{\text{samp}}) = \sqrt{\frac{1}{|\mathcal{B}_G(S)|} \sum_{(i,j)\in\mathcal{B}_G(S)} \left(\widehat{p}_{ij}^{(N_{\text{samp}})} - \widehat{p}_{ij}^{(100000)}\right)^2}.$$

The RMSD decreased consistently with increasing sample count and followed the expected Monte Carlo rate of approximately $N_{\text{samp}}^{-1/2}$ (Fig. 5b) [4]. The observed RMSD was 0.0027 at 1000 samples and 0.001 at 10000 samples, indicating that 1000 samples already provided accurate posterior estimates for the downstream summaries considered here.

## 4.4 RNA shapes can recover pseudoknotted motifs missed by centroid decoding

The sampled structures produced by PRISM can be summarized either at the level of exact base pairs or at the level of RNA shapes. The centroid structure is a base-pair-level posterior decoder: a base pair contributes positively to the centroid objective only when its posterior probability exceeds 1/2. This makes the centroid useful for identifying well-supported individual contacts, but it can miss structural motifs whose base pairs shift across nearby positions. In particular, if a pseudoknotted helix occurs in most sampled structures but its

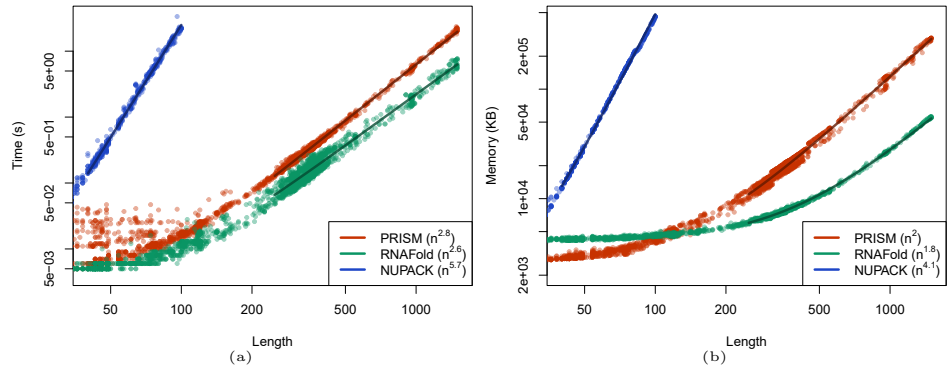



**Figure 4** Runtime and memory usage of PRISM, RNAFold, and NUPACK 3.0 on the benchmark dataset with structure constraints. Runtime and memory are shown as functions of input length on log–log axes. Empirical scaling was estimated by fitting power laws for lengths $n \geq 250$, except for NUPACK 3.0, where fitting was performed for $n \geq 40$. Fitted trends are shown as solid lines, and the corresponding exponents $x$ are reported in the legend as empirical complexities of the form $\Theta(n^x)$.

endpoints vary slightly, the posterior mass can be distributed among nearby crossing base pairs. Then no single crossing base pair exceeds the centroid threshold, even though the pseudoknotted motif is common in the ensemble.

RNA-shape summaries address this issue by grouping structures according to topology rather than exact base-pair positions. The following example shows a sequence, its MFE structure, its centroid structure, and the two most frequent level-5 shapes observed among sampled structures:

(1) `GGGUGGUGUGUACCACCCCUGAUGAGUCCGAAAGGACGAA` sequence,
(2) `(((((((.[[[).)))))))...]]].((((....))))...` MFE structure,
(3) `(((((((....)))))))........((((....))))...` centroid structure,
(4) `[][]()` (0.806), `[][]()()` (0.175) top two shapes.

The MFE structure contains crossing base pairs, whereas the centroid structure does not. However, the dominant RNA shape, `[][]()`, has estimated posterior frequency 0.806, showing that the crossing topology is present in the majority of sampled structures. The discrepancy occurs because the pseudoknotted base pairs are shifted across nearby positions in the sample, preventing any single crossing pair from exceeding the centroid threshold. Thus, shape-level posterior summaries can reveal pseudoknotted ensemble features that are hidden from exact base-pair-level centroid decoding.

## 4.5 RNA-shape distributions reveal alternative conformations in the ZMP–ZTP family

We used the ZMP–ZTP riboswitch family [31], Rfam identifier RF01750, as a test case for evaluating whether RNA-shape distributions can summarize alternative conformations across a structured RNA family. RF01750 is a particularly informative example because the annotated aptamer contains a conserved pseudoknotted fold [24, 29], whereas the full riboswitch is thought to regulate transcription through competition between the ligand-stabilized aptamer and alternative, pseudoknot-disrupting expression-platform structures [27]. Thus, this family provides a natural setting in which to ask whether shape distributions recover both the conserved pseudoknotted architecture and plausible pseudoknot-free alternatives.

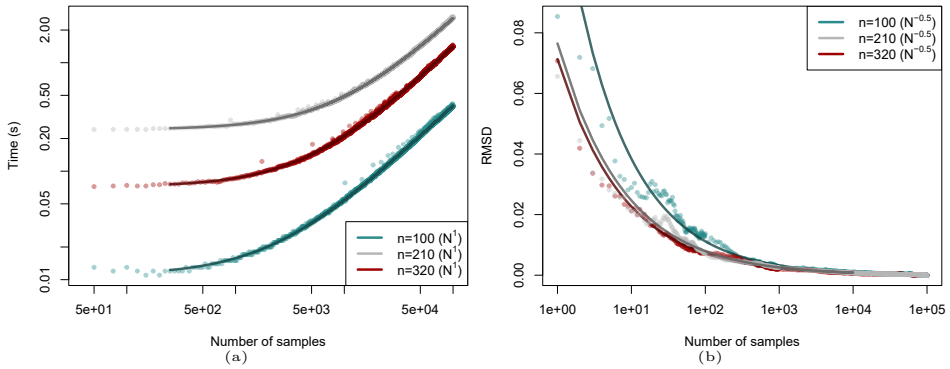



**Figure 5** Effect of stochastic traceback sample count on PRISM runtime and posterior probability estimates. (a) Runtime as a function of the number of samples for three sequence lengths. Increasing the number of samples adds linear overhead after the dynamic-programming tables have been computed. (b) RMSD of estimated base-pair probabilities relative to estimates obtained from 100000 samples. The fitted exponents are close to $-1/2$, matching the standard Monte Carlo convergence rate for estimating Bernoulli probabilities.

As described in Section 3, we decomposed the Rfam consensus structure into two partial input structures. The first, $G_{\text{big}}$, is the maximal pseudoknot-free component obtained after removing crossing base pairs. The second, $G_{\text{small}}$, contains the removed crossing component. Each partial structure was fitted to each ungapped RF01750 sequence and then used as an input constraint for PRISM. For every sequence/input pair, we generated stochastic traceback samples, mapped each sampled structure to its level-5 RNA shape, and retained at most the five most frequent shapes whose estimated posterior frequency exceeded 0.1. Shapes below this threshold were omitted.

Figure 6 shows the resulting shape distributions. Each point represents one retained shape for one sequence/input pair. Circles and triangles distinguish samples generated from $G_{\text{big}}$ and $G_{\text{small}}$, respectively. Point color gives the rank of the shape within that sequence/input pair, with rank 1 corresponding to the highest estimated posterior frequency. The sequences are ordered by length on the $y$-axis, and the horizontal reference line marks the transition near length 95. The dendrogram above the $x$-axis clusters the observed level-5 shapes; shapes without square brackets correspond to pseudoknot-free classes, whereas shapes containing square brackets retain a crossing, pseudoknotted component.

For sequences of length at most 95, the highest-ranked shapes under the two input constraints are concentrated around the consensus pseudoknotted shape, `([()(])`, shown in bold in Figure 6. This indicates that, for shorter RF01750 sequences, both the pseudoknot-free component and the removed crossing component provide enough information for PRISM to recover the same family-level architecture. In this length range, the two partial constraints therefore lead to compatible posterior shape distributions rather than to distinct dominant folds.

For longer sequences, the distributions become more heterogeneous and depend more strongly on the input constraint. Under $G_{\text{small}}$, many sequences shift toward the pseudoknotted shape `([(()()))(])`, a variant of the consensus shape with an additional nested loop, while pseudoknot-free shapes such as `(()())` remain recurrent. Under $G_{\text{big}}$, the sampled structures are spread across several pseudoknot-containing and pseudoknot-free classes, including shapes in which the crossing component appears in a different shape context or is absent from the retained high-probability class. Thus, the longer sequences show greater shape-level plasticity: the pseudoknotted component is not lost uniformly, but its placement and compatibility with surrounding helices vary across the family.

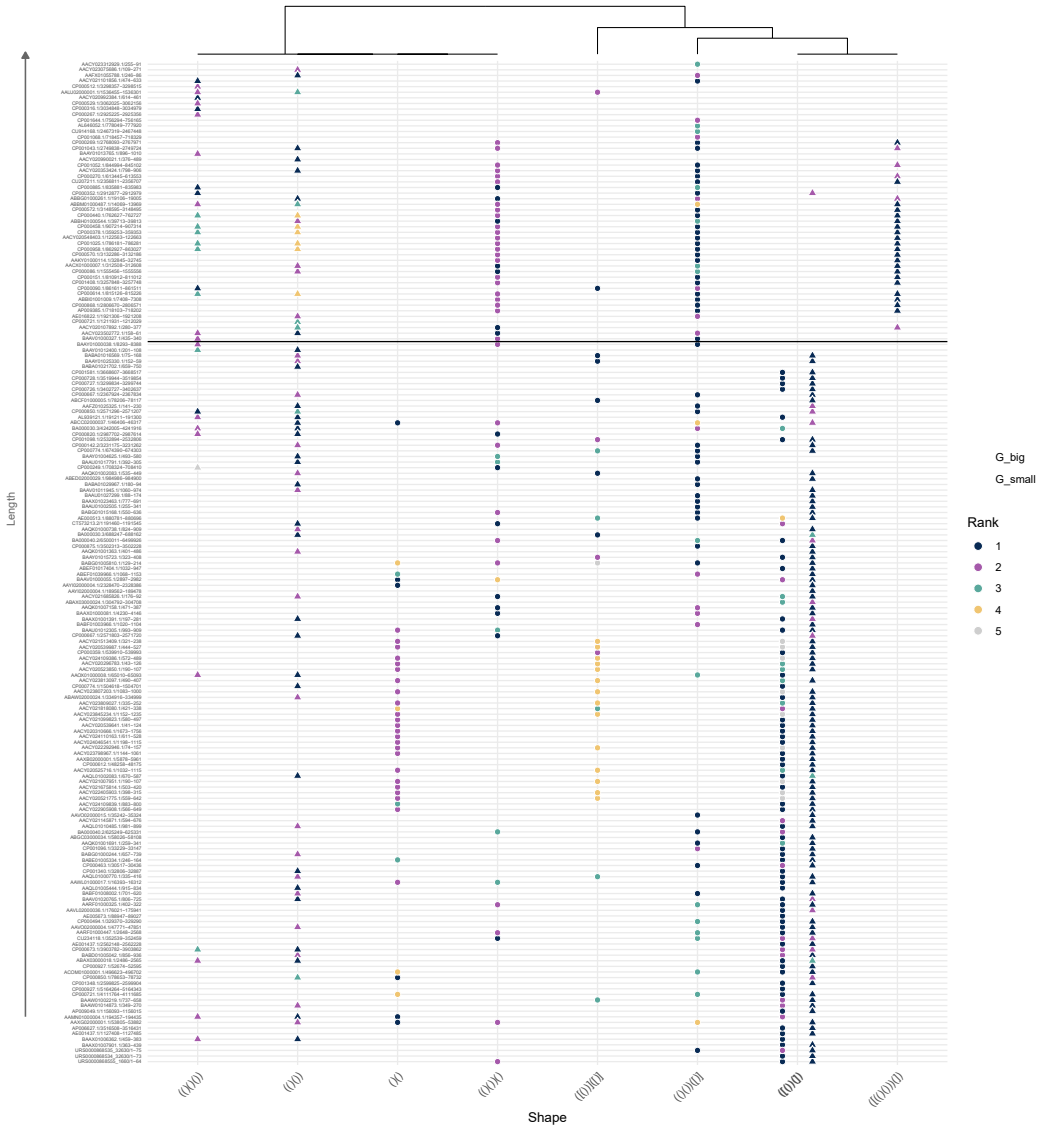



**Figure 6** Distribution of level-5 RNA shapes for the ZMP–ZTP riboswitch family RF01750. For each sequence and input constraint, the five most frequent shapes are shown. The x-axis gives the RNA shape, and the y-axis lists sequences ordered by length. A horizontal line marks the boundary at sequence length 95. Point color indicates the estimated posterior frequency of the shape among sampled structures, and point shape distinguishes the two input constraints $G_{\text{big}}$ and $G_{\text{small}}$. Points with $p < 0.1$ are excluded.

These results illustrate the value of RNA-shape distributions for family-level analysis. A single decoded structure would obscure the coexistence of consensus-like pseudoknotted structures, pseudoknot-free alternatives, and pseudoknotted variants with additional loops. By contrast, the distributional view reveals how these alternatives depend on both sequence length and the partial structural constraint used as input. We interpret the pseudoknot-free classes as computationally recovered alternatives that are consistent with the known regulatory competition between aptamer and expression-platform structures, not as evidence for a second ligand-stabilized aptamer fold. In the biological ZMP–ZTP riboswitch, ligand binding is expected to stabilize the pseudoknotted aptamer state; the pseudoknot-free conformations are more naturally interpreted as apo-compatible, terminator-compatible, or off-pathway alternatives.

## 5 Discussion and Future Directions

PRISM substantially extends the practical value of efficient hierarchical partition function computation over our previous algorithm CParty. Where CParty computes only a scalar ensemble quantity, we obtain posterior probabilities of structural features and structural classes in PRISM. Our empirical results demonstrate original ensemble-based analysis of pseudoknotted RNAs, as enabled by PRISM's novel functionality. CParty's algorithm efficiently computes $Z(S \mid G)$; as a side effect of efficient computation by dynamic programming, it additionally computes partition functions of subproblems. In turn, PRISM utilizes this information to estimate posterior masses for interpretable classes, including base-pair events, unpaired-nucleotide events, pseudoknot-conditioned subensembles, decoded structures, and RNA-shape abstractions. This class-decomposition view is the main conceptual distinction between CParty and PRISM: CParty computes the constrained partition function, whereas PRISM separates its Boltzmann mass into summaries that are directly useful for structural interpretation.

The validation experiments support the correctness of this posterior-analysis layer. When $G = \emptyset$, the conditional ensemble reduces to the standard pseudoknot-free secondary-structure ensemble, and PRISM reproduces RNAFold values for MFE energy, ensemble free energy, centroid expected distance, and MEA score under the same energy parameters. The comparison between the Boltzmann probability of the MFE structure and its observed frequency in stochastic traceback samples further supports that the traceback procedure samples from the intended conditional Boltzmann distribution.

The computational experiments show that the posterior-analysis layer remains practical over the tested sequence lengths. The full workflow, including computation of $Z(S \mid G)$, stochastic traceback, centroid decoding, and MEA decoding, remains close to RNAFold in empirical scaling and below NUPACK 3.0 over the shared benchmark range. Sampling introduces a transparent tradeoff: increasing the number of samples increases runtime linearly, while the RMSD of estimated base-pair probabilities decreases at approximately the standard Monte Carlo rate. This behavior makes the sample count an explicit accuracy parameter. In our experiments, 1000 samples gave small RMSD relative to estimates obtained from 100000 samples, suggesting that moderate sample sizes are sufficient for many posterior-decoding tasks.

The results also show that base-pair-level summaries and topology-level summaries answer different questions. Centroid decoding is appropriate when the same exact contacts recur with high probability, but it can miss a pseudoknotted motif whose helix endpoints shift across sampled structures. The pseudoknot-conditioned centroid reduces dilution from pseudoknot-

free samples, but it still operates at the level of exact base-pair identities. RNA-shape summaries provide a complementary view by grouping structures according to topology. In the example from Section 4, the centroid structure did not contain crossing base pairs, yet the dominant level-5 shape was pseudoknotted and appeared with high posterior frequency. This illustrates why posterior mass can be weak at the level of individual shifted base pairs but strong at the level of structural classes. The ZMP–ZTP riboswitch [31] analysis demonstrates how shape distributions can be used to compare conditional ensembles across related sequences and input constraints. By fitting $G_{\text{big}}$ and $G_{\text{small}}$ to each ungapped sequence and computing the most frequent level-5 shapes, PRISM summarizes how the posterior distribution over topologies changes with sequence length and with the chosen input scaffold. For shorter sequences, both inputs frequently recover the consensus-like shape. For longer sequences, the dominant shapes diverge: one input favors variants with additional loops, while the other shows changes in the placement of the pseudoknotted component or shifts toward pseudoknot-free conformations. These results suggest that shape-level posterior summaries can help identify alternative conformational classes within an RNA family, especially when single decoded structures are too coarse to represent ensemble variation.

Several limitations remain. First, PRISM analyzes a conditional ensemble defined by a fixed pseudoknot-free input structure $G$. The resulting posterior probabilities are therefore conditional on the chosen scaffold. If $G$ is inaccurate or if the biological system occupies multiple competing scaffolds, then a single $\Omega[S, G]$ may not capture all relevant alternatives. Second, the admissible ensemble remains restricted to hierarchically generated density-2 structures. This class is broad enough to include many important pseudoknotted motifs, but it does not represent arbitrary pseudoknots. Third, base-pair probabilities and shape frequencies are estimated by stochastic traceback rather than computed exactly. The sampling experiments show predictable Monte Carlo convergence, but rare structural features may require larger sample sizes. Finally, the level-5 shape abstraction intentionally removes positional details. This is useful for detecting conserved topology, but it can merge structures that differ in functionally relevant loop lengths or exact contact positions.

These limitations suggest natural extensions. One direction is to develop adaptive sampling procedures that continue stochastic traceback until base-pair probabilities or shape frequencies reach a specified confidence threshold. Another is to compute selected class-specific partition-function masses more directly, rather than estimating all class probabilities by sampling. A third direction is to integrate external evidence, such as comparative information or probing-derived constraints, into the choice of $G$ or into posterior reweighting of sampled structures. Finally, shape-aware decoders could return representative structures from high-probability shape classes, combining the interpretability of a concrete structure with the robustness of topology-level posterior analysis.

## References


1 Tatsuya Akutsu. Dynamic programming algorithms for RNA secondary structure prediction with pseudoknots. *Discrete Applied Mathematics*, 104(1-3):45–62, 2000. doi:10.1016/S0166-218X(00)00186-4.

2 Mirela Andronescu, Vera Bereg, Holger H. Hoos, and Anne Condon. RNA STRAND: the RNA secondary structure and statistical analysis database. *BMC Bioinformatics*, 9:340, 2008. doi:10.1186/1471-2105-9-340.

3 Mirela S. Andronescu, C. Pop, and Anne E. Condon. Improved free energy parameters for RNA pseudoknotted secondary structure prediction. *RNA*, 16(1):26–42, 2010.

4 Robert P Christian and George Casella. *Monte Carlo Statistical Methods*. Springer Texts in Statistics, 2004.

5 Jose Almeida Cruz and Eric Westhof. The dynamic landscapes of RNA architecture. *Cell*, 136(4):604–609, 2009. doi:10.1016/j.cell.2009.02.003.

6 Ye Ding, Chi Yu Chan, and Charles E. Lawrence. RNA secondary structure prediction by centroids in a Boltzmann weighted ensemble. *RNA*, 11(8):1157–1166, 2005. doi:10.1261/rna.2500605.

7 Ye Ding and Charles E Lawrence. A statistical sampling algorithm for RNA secondary structure prediction. *Nucleic Acids Research*, 31:7280–7301, December 2003.

8 Chuong B. Do, Daniel A. Woods, and Serafim Batzoglou. CONTRAfold: RNA secondary structure prediction without physics-based models. In *Proceedings of the Tenth Annual International Conference on Research in Computational Molecular Biology*, pages 90–98, 2006. doi:10.1093/bioinformatics/btl246.

9 Mateo Gray, Luke Trinity, Ulrike Stege, Yann Ponty, Sebastian Will, and Hosna Jabbari. CParty: hierarchically constrained partition function of RNA pseudoknots. *Bioinformatics*, 41(1):btae748, 2025. doi:10.1093/bioinformatics/btae748.

10 Mateo Gray, Sebastian Will, and Hosna Jabbari. PRISM. Software, swhId: swh:1:dir:79803e70b92d8457ba08d16fe67b09382501716c (visited on 2026-08-13). URL: https://github.com/TheCOBRALab/PRISM, doi:10.4230/artifacts.27651.

11 Mateo Gray, Sebastian Will, and Hosna Jabbari. PRISM-RawData. Dataset, swhId: swh:1:dir:4646c334d25f499a6485f62fe913bb90e2cdd115 (visited on 2026-08-13). URL: https://github.com/mateog4712/PRISM-RawData, doi:10.4230/artifacts.27652.

12 Sam Griffiths-Jones, Simon Moxon, Mhairi Marshall, Ajay Khanna, Sean R Eddy, and Alex Bateman. Rfam: annotating non-coding rnas in complete genomes. *Nucleic Acids Research*, 33(suppl_1):D121–D124, 2005.

13 Hosna Jabbari, Anne Condon, Ana Pop, Cristina Pop, and Yinglei Zhao. Hfold: Rna pseudoknotted secondary structure prediction using hierarchical folding. In Raffaele Giancarlo and Sridhar Hannenhalli, editors, *Algorithms in Bioinformatics*, pages 323–334, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. doi:10.1007/978-3-540-74126-8_30.

14 Hosna Jabbari, Anne Condon, and Shelly Zhao. Novel and Efficient RNA Secondary Structure Prediction Using Hierarchical Folding. *Journal of Computational Biology*, 15(2):139–163, March 2008. doi:10.1089/cmb.2007.0198.

15 Hosna Jabbari, Ian Wark, Carlo Montemagno, and Sebastian Will. Sparsification Enables Predicting Kissing Hairpin Pseudoknot Structures of Long RNAs in Practice. In *17th International Workshop on Algorithms in Bioinformatics (WABI 2017)*, volume 88 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.WABI.2017.12.

16 Hosna Jabbari, Ian Wark, Carlo Montemagno, and Sebastian Will. Knotty: efficient and accurate prediction of complex RNA pseudoknot structures. *Bioinformatics*, 34:3849–3856, November 2018. doi:10.1093/bioinformatics/bty420.

17 Marilyn Kozak. Regulation of translation via mRNA structure in prokaryotes and eukaryotes. *Gene*, 361:13–37, 2005. doi:10.1016/j.gene.2005.06.037.

18 Ronny Lorenz, Stephan H Bernhart, Christian Höner zu Siederdissen, Hakim Tafer, Christoph Flamm, Peter F Stadler, and Ivo L Hofacker. Viennarna package 2.0. *Algorithms for Molecular Biology*, 6:1–14, 2011.

19 Rune B Lyngsø and Christian NS Pedersen. Pseudoknots in RNA secondary structures. In *Proceedings of the fourth annual international conference on Computational molecular biology*, pages 201–209, New York, NY, United States, 2000. Association for Computing Machinery.

20 John S. McCaskill. The equilibrium partition function and base pair binding probabilities for RNA secondary structure. *Biopolymers*, 29(6-7):1105–1119, 1990.

21 Stephanie A. Mortimer, Mary Anne Kidwell, and Jennifer A. Doudna. Insights into RNA structure and function from genome-wide studies. *Nature Reviews Genetics*, 15:469–479, 2014. doi:10.1038/nrg3681.

22 Yann Ponty. Efficient sampling of RNA secondary structures from the Boltzmann ensemble of low-energy. *Journal of Mathematical Biology*, 56:107–127, 2008. doi:10.1007/s00285-007-0137-z.

23 Christian M. Reidys, Fenix W. D. Huang, Jørgen E. Andersen, Robert C. Penner, Peter F. Stadler, and Markus E. Nebel. Topology and prediction of RNA pseudoknots. *Bioinformatics*, 27(8):1076–1085, 2011. doi:10.1093/bioinformatics/btr090.

24 Aiming Ren, Kanagalaghatta R Rajashankar, and Dinshaw J Patel. Global rna fold and molecular recognition for a pfl riboswitch bound to zmp, a master regulator of one-carbon metabolism. *Structure*, 23:1375–1381, 2015. doi:10.1016/j.str.2015.05.016.

25 Saad Sheikh, Rolf Backofen, and Yann Ponty. Impact of the energy model on the complexity of rna folding with pseudoknots. In *Annual Symposium on Combinatorial Pattern Matching*, pages 321–333. Springer, 2012. doi:10.1007/978-3-642-31265-6_26.

26 Peter Steffen, Björn Voß, Marc Rehmsmeier, Jens Reeder, and Robert Giegerich. RNAshapes: an integrated RNA analysis package based on abstract shapes. *Bioinformatics*, 22(4):500–503, 2006. doi:10.1093/bioinformatics/btk010.

27 Eric J Strobel, Luyi Cheng, Katherine E Berman, Paul D Carlson, and Julius B Lucks. A ligand-gated strand displacement mechanism for ztp riboswitch transcription control. *Nature Chemical Biology*, 15:1067–1076, 2019. doi:10.1038/s41589-019-0382-7.

28 Ignacio Tinoco Jr and Carlos Bustamante. How RNA folds. *Journal of Molecular Biology*, 293(2):271–281, 1999.

29 Jeremiah J Trausch, Joan G Marcano-Velázquez, Michal M Matyjasik, and Robert T Batey. Metal ion-mediated nucleobase recognition by the ztp riboswitch. *Chemistry and Biology*, 22:829–837, 2015. doi:10.1016/j.chembiol.2015.06.007.

30 M. Bryan Warf and J. Andrew Berglund. Role of RNA structure in regulating pre-mRNA splicing. *Trends in Biochemical Sciences*, 35(3):169–178, 2010. doi:10.1016/j.tibs.2009.10.004.

31 Zasha Weinberg, Joy X Wang, Jarrod Bogue, Jingying Yang, Keith Corbino, Ryan H Moy, and Ronald R Breaker. Comparative genomics reveals 104 candidate structured rnas from bacteria, archaea, and their metagenomes. *Genome Biology*, 11, 2010. doi:10.1186/gb-2010-11-3-r31.

32 Timothy J. Wilson and David M. J. Lilley. RNA catalysis—is that it? *RNA*, 21(4):534–537, 2015. doi:10.1261/rna.049874.115.

33 Christina Witwer, Ivo L Hofacker, and Peter F Stadler. Prediction of consensus RNA secondary structures including pseudoknots. *IEEE/ACM Transactions in Computational Biology and Bioinformatics*, 1(2):66–77, 2004. doi:10.1109/TCBB.2004.22.

34 Joseph N Zadeh, Conrad D Steenberg, Justin S Bois, Brian R Wolfe, Marshall B Pierce, Asif R Khan, Robert M Dirks, and Niles A Pierce. NUPACK: analysis and design of nucleic acid systems. *Journal of Computational Chemistry*, 32(1):170–173, 2011. doi:10.1002/JCC.21596.

# Minimum Flow Decomposition Guided by Saturating Subflows

**Ke Chen**[1]
Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA, USA

**Abhishek Talesara**
Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA, USA

**Sanchal Thakkar**
Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA, USA

**Mingfu Shao**[1]
Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA, USA
Huck Institutes of the Life Sciences, The Pennsylvania State University, University Park, PA, USA

## Abstract

**Introduction.** The minimum flow decomposition (MFD) problem asks to decompose a directed acyclic flow network $(G, f)$ with a unique source $s$ and a unique sink $t$ into the fewest weighted $s$-$t$ paths whose combined contributions exactly reproduce $f$. MFD underlies a broad class of multi-assembly tasks in bioinformatics: reference-based RNA assembly from splice graphs [24, 11, 23, 22, 16, 18, 15, 19, 29, 25], metagenomic assembly [21], and viral quasi-species inference [1].

MFD is strongly NP-hard [26] and hard to approximate within some fixed constant factor [12]. Exact solvers include an FPT algorithm whose runtime grows exponentially in the solution size [14] and a family of integer linear programming (ILP) formulations [5, 9] capable of handling extensions such as inexact flows [27, 4], safety and subpath constraints [28, 7, 13, 3], and graphs with cycles [6]. However, ILP remains unscalable on large practical instances.

The widely used greedy-width heuristic [26] is very efficient but can be exponentially worse than optimal in the worst case [2]. The state-of-the-art heuristic, catfish [20], substantially improves this efficiency-performance tradeoff by identifying linear equations among edge flow values – structural constraints implied by any optimal decomposition – and resolving them via safe graph transformations. On simpler instances catfish is highly effective, but three interrelated limitations degrade its performance on complex graphs: (1) it cannot distinguish *good* equations (arising from a true optimal decomposition) from *superficial* ones that distort the graph when resolved; (2) many good equations cannot be fully resolved due to the absence of suitable closed subgraphs, so catfish discards their information entirely; and (3) when no equation is resolvable catfish falls back to greedy-width, which performs poorly on entangled graphs.

**Method.** We introduce catfish-LP, which augments catfish with a lightweight linear programming (LP) formulation based on *saturating subflows*. For each edge $e \in E$ we define continuous variables $\{x_e(a) : a \in E\}$ modeling a valid $s$-$t$ subflow that saturates $e$; intuitively, $x_e(a)$ represents the amount of flow on $e$ that must passes through edge $a$. Five base constraints enforce saturation, symmetry, flow validity, and flow conservation. Two additional equation constraints require that the aggregate subflow through the left-hand-side edges of an equation equals that through the right-hand-side edges. The full LP is polynomial-time solvable, adding only modest overhead over catfish.

The LP plays three complementary roles within a single unified framework: (1) *equation filtering*: if adding a candidate equation renders the LP infeasible, that equation cannot arise from any minimum decomposition and is discarded, preventing structurally invalid graph transformations;

[1] Corresponding authors.

© Ke Chen, Abhishek Talesara, Sanchal Thakkar, and Mingfu Shao;
licensed under Creative Commons License CC-BY 4.0
26th International Conference on Algorithms for Bioinformatics (WABI 2026).
Editors: Nadia El-Mabrouk and Fabio Vandin; Article No. 31; pp. 31:1–31:5
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

(2) *safe edge merging*: a feasible LP solution reveals pairs of edges that must carry identical subflow and can therefore be safely contracted; (3) *informed greedy extraction*: when no further simplification is possible, rather than invoking greedy-width blindly, catfish-LP extracts from the LP solution the heaviest simple path consistent with all surviving equations, deferring the error-prone greedy step as long as possible.

**Experimental Results.** We compare catfish-LP against greedy-width, catfish, and the optimized ILP solver [9] on two benchmarks, using Gurobi [10] as the underlying LP/ILP engine. Table 1 reports decomposition quality on four datasets of biologically derived splice graphs, with abundances estimated by Salmon [17] or simulated with the Flux-Simulator [8]. Catfish-LP achieves the smallest excess among heuristics on the Salmon dataset and negative excess on the remaining three, matching ILP quality while being orders of magnitude faster. Figure 1 summarizes results on 1,440 simulated graphs spanning 72 complexity configurations. Catfish-LP consistently produces the smallest decompositions and recovers the most ground-truth paths among all heuristics, achieving near-ILP quality in a fraction of its runtime – including a threefold improvement over catfish on the hardest 498 instances where ILP times out on every instance.

**Conclusion.** Catfish-LP demonstrates that incorporating a polynomial-time LP oracle into a combinatorial heuristic yields substantial gains in decomposition quality with negligible scalability cost, addressing each of catfish's core limitations in a unified manner. Future directions include strengthened LP formulations, domain-specific constraints for transcriptome assembly, and probabilistic interpretations of LP-guided decompositions.

**2012 ACM Subject Classification** Applied computing → Computational transcriptomics; Applied computing → Bioinformatics; Mathematics of computing → Network flows

**Keywords and phrases** flow decomposition, RNA assembly, linear programming, optimality gap

**Digital Object Identifier** 10.4230/LIPIcs.WABI.2026.31

**Category** Extended Abstract

**Related Version** *Full Version*: `https://doi.org/10.64898/2025.12.11.693570`

**Supplementary Material** *Software (Source Code)*: `https://github.com/Shao-Group/catfish-LP` archived at `swh:1:dir:fb27825b2bba25a8f04b1ae7ff47369df018a4d3`

**Funding** This work is supported by the US National Science Foundation (2145171 to M.S.) and by the US National Institutes of Health (R01HG011065 to M.S.).

**Table 1** Decomposition results on perfect splice graphs (metric: $|P| - |P^\dagger|$, lower is better). $|P^\dagger|$ is the total number of ground-truth RNA isoforms, $|P|$ is the total decomposition size. For ILP, the parenthesised number is the ground-truth path count in timed-out instances.

| Dataset | Graphs | $\lvert P^\dagger\rvert$ | $\lvert P\rvert - \lvert P^\dagger\rvert$ | | | |
|---|---|---|---|---|---|---|
| | | | greedy-width | catfish | catfish-LP | ILP |
| Salmon | 13,300,893 | 32,889,145 | 1,330,137 | 86,355 | 15,093 | −11,072 (43,476) |
| human | 1,169,083 | 2,139,559 | 13,252 | 491 | −142 | −763 (169) |
| mouse | 1,316,058 | 2,142,500 | 11,410 | −1 | −817 | −1,332 (233) |
| zebrafish | 1,549,373 | 2,140,200 | 2,928 | −387 | −386 | −508 (16) |

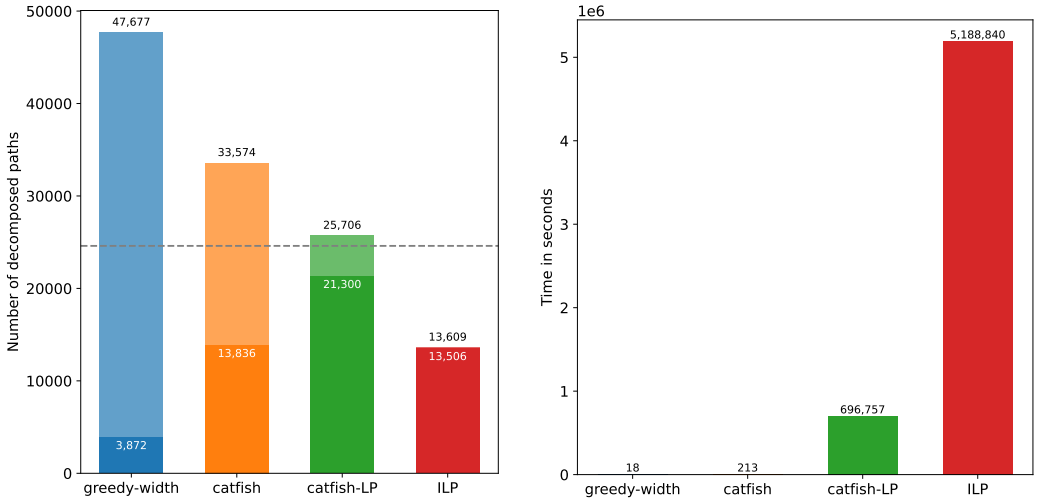



**Figure 1** Decomposition quality and runtime on the full simulated dataset (1,440 graphs across 72 configurations). In the left subfigure, the dashed horizontal line marks the total number of ground-truth paths across all instances ($|P^\dagger| = 24,600$). For each method, the full bar represents the total number of paths produced, while the shaded portion denotes the number of decomposed paths that exactly match a ground-truth path. The ILP bar is heavily skewed because ILP failed to decompose 498 graph instances (35%) within the time limit.

## References


1 Jasmijn A Baaijens, Leen Stougie, and Alexander Schönhuth. Strain-aware assembly of genomes from mixed samples using flow variation graphs. In Russell Schwartz, editor, *International Conference on Research in Computational Molecular Biology*, Lecture Notes in Computer Science, pages 221–222. Springer, 2020. doi:10.1007/978-3-030-45257-5_14.

2 Manuel Cáceres, Massimo Cairo, Andreas Grigorjew, Shahbaz Khan, Brendan Mumey, Romeo Rizzi, Alexandru I Tomescu, and Lucia Williams. Width helps and hinders splitting flows. *ACM Transactions on Algorithms*, 20(2):1–20, 2024. doi:10.1145/3641820.

3 Fernando HC Dias, Manuel Cáceres, Lucia Williams, Brendan Mumey, and Alexandru I Tomescu. A safety framework for flow decomposition problems via integer linear programming. *Bioinformatics*, 39:btad640, 2023. doi:10.1093/bioinformatics/btad640.

4 Fernando HC Dias and Alexandru I Tomescu. Accurate flow decomposition via robust integer linear programming. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 21:1955–1964, 2024. doi:10.1109/TCBB.2024.3433523.

5 Fernando HC Dias, Lucia Williams, Brendan Mumey, and Alexandru I Tomescu. Efficient minimum flow decomposition via integer linear programming. *Journal of Computational Biology*, 29(11):1252–1267, 2022. doi:10.1089/cmb.2022.0257.

6 Fernando HC Dias, Lucia Williams, Brendan Mumey, and Alexandru I Tomescu. Minimum flow decomposition in graphs with cycles using integer linear programming. *Journal of Global Optimization*, 93:1145–1176, 2025. doi:10.1007/s10898-025-01556-8.

7 Daniel Gibney, V Sharma, Srinivas Aluru, et al. Feasibility of flow decomposition with subpath constraints in linear time. In Christina Boucher and Sven Rahmann, editors, *22nd International Workshop on Algorithms in Bioinformatics (WABI 2022)*, volume 242 of *LIPIcs*, pages 17:1–17:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.WABI.2022.17.

8 Thasso Griebel, Benedikt Zacher, Paolo Ribeca, Emanuele Raineri, Vincent Lacroix, Roderic Guigó, and Michael Sammeth. Modelling and simulating generic RNA-Seq experiments with the flux simulator. *Nucleic acids research*, 40(20):10073–10083, 2012.

9 Andreas Grigorjew, Fernando HC Dias, Andrea Cracco, Romeo Rizzi, and Alexandru I Tomescu. Accelerating ILP solvers for minimum flow decompositions through search space and dimensionality reductions. In Leo Liberti, editor, *22nd International Symposium on Experimental Algorithms (SEA 2024)*, volume 301 of *LIPIcs*, pages 14:1–14:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.SEA.2024.14.

10 Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2024. URL: https://www.gurobi.com.

11 Mitchell Guttman, Manuel Garber, Joshua Z Levin, Julie Donaghey, James Robinson, Xian Adiconis, Lin Fan, Magdalena J Koziol, Andreas Gnirke, Chad Nusbaum, et al. Ab initio reconstruction of cell type-specific transcriptomes in mouse reveals the conserved multi-exonic structure of lincRNAs. *Nature Biotechnology*, 28(5):503–510, 2010.

12 Tzvika Hartman, Avinatan Hassidim, Haim Kaplan, Danny Raz, and Michal Segalov. How to split a flow? In Albert G. Greenberg and Kazem Sohraby, editors, *2012 Proceedings IEEE INFOCOM*, pages 828–836. IEEE, IEEE, 2012. doi:10.1109/INFCOM.2012.6195830.

13 Shahbaz Khan, Milla Kortelainen, Manuel Cáceres, Lucia Williams, and Alexandru I Tomescu. Improving RNA assembly via safety and completeness in flow decompositions. *Journal of Computational Biology*, 29(12):1270–1287, 2022. doi:10.1089/cmb.2022.0261.

14 Kyle Kloster, Philipp Kuinke, Michael P O'Brien, Felix Reidl, Fernando Sánchez Villaamil, Blair D Sullivan, and Andrew van der Poel. A practical FPT algorithm for flow decomposition and transcript assembly. In Rasmus Pagh and Suresh Venkatasubramanian, editors, *2018 Proceedings of the Twentieth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 75–86. SIAM, SIAM, 2018. doi:10.1137/1.9781611975055.7.

15 Sam Kovaka, Aleksey V Zimin, Geo M Pertea, Roham Razaghi, Steven L Salzberg, and Mihaela Pertea. Transcriptome assembly from long-read RNA-Seq alignments with StringTie2. *Genome Biology*, 20:1–13, 2019.

16 Juntao Liu, Ting Yu, Tao Jiang, and Guojun Li. TransComb: genome-guided transcriptome assembly via combing junctions in splicing graphs. *Genome Biology*, 17:1–9, 2016.

17 Rob Patro, Geet Duggal, Michael I Love, Rafael A Irizarry, and Carl Kingsford. Salmon provides fast and bias-aware quantification of transcript expression. *Nature methods*, 14(4):417–419, 2017.

18 Mihaela Pertea, Geo M Pertea, Corina M Antonescu, Tsung-Cheng Chang, Joshua T Mendell, and Steven L Salzberg. StringTie enables improved reconstruction of a transcriptome from RNA-Seq reads. *Nature Biotechnology*, 33(3):290–295, 2015.

19 Mingfu Shao and Carl Kingsford. Accurate assembly of transcripts through phase-preserving graph decomposition. *Nature Biotechnology*, 35(12):1167–1169, 2017.

20 Mingfu Shao and Carl Kingsford. Theory and a heuristic for the minimum path flow decomposition problem. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 16(2):658–670, 2017. doi:10.1109/TCBB.2017.2779509.

21 Jim Shaw, Jean-Sebastien Gounot, Hanrong Chen, Niranjan Nagarajan, and Yun William Yu. Floria: fast and accurate strain haplotyping in metagenomes. *Bioinformatics*, 40(Supplement_1):i30–i38, 2024. doi:10.1093/bioinformatics/btae252.

22 Li Song, Sarven Sabunciyan, and Liliana Florea. CLASS2: accurate and efficient splice variant annotation from RNA-Seq reads. *Nucleic Acids Research*, 44(10):e98–e98, 2016.

23 Alexandru I Tomescu, Anna Kuosmanen, Romeo Rizzi, and Veli Mäkinen. A novel min-cost flow method for estimating transcript expression with RNA-Seq. *BMC Bioinformatics*, 14(S15):1–10, 2013. doi:10.1186/1471-2105-14-S5-S15.

24 Cole Trapnell, Brian A Williams, Geo Pertea, Ali Mortazavi, Gordon Kwan, Marijke J Van Baren, Steven L Salzberg, Barbara J Wold, and Lior Pachter. Transcript assembly and quantification by rNA-Seq reveals unannotated transcripts and isoform switching during cell differentiation. *Nature Biotechnology*, 28(5):511–515, 2010.

**25** Laura H Tung, Mingfu Shao, and Carl Kingsford. Quantifying the benefit offered by transcript assembly with Scallop-LR on single-molecule long reads. *Genome Biology*, 20:1–18, 2019.

**26** Benedicte Vatinlen, Fabrice Chauvet, Philippe Chrétienne, and Philippe Mahey. Simple bounds and greedy algorithms for decomposing a flow into a minimal set of paths. *European Journal of Operational Research*, 185(3):1390–1401, 2008. doi:10.1016/j.ejor.2006.05.043.

**27** Lucia Williams, Gillian Reynolds, and Brendan Mumey. Rna transcript assembly using inexact flows. In Illhoi Yoo, Jinbo Bi, and Xiaohua Hu, editors, *2019 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 1907–1914. IEEE, IEEE, 2019. doi:10.1109/BIBM47256.2019.8983180.

**28** Lucia Williams, Alexandru I Tomescu, and Brendan Mumey. Flow decomposition with subpath constraints. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 20(1):360–370, 2022. doi:10.1109/TCBB.2022.3147697.

**29** Qimin Zhang, Qian Shi, and Mingfu Shao. Accurate assembly of multi-end RNA-Seq data with Scallop2. *Nature Computational Science*, 2(3):148–152, 2022. doi:10.1038/s43588-022-00216-1.