

# 27th International Conference on Database Theory

ICDT 2024, March 25–28, 2024, Paestum, Italy

Edited by

Graham Cormode

Michael Shekelyan



#### Editors

**Graham Cormode** 

University of Warwick, UK  
g.cormode@warwick.ac.uk

**Michael Shekelyan** 

Queen Mary University of London, UK  
m.shekelyan@qmul.ac.uk

#### ACM Classification 2012

Information systems → Data management systems; Information systems → Structured Query Language; Information systems → Relational database query languages; Information systems → Information extraction; Information systems → Join algorithms; Information systems → Query languages; Information systems → Relational database model; Theory of computation; Theory of computation → Database theory; Theory of computation → Logic; Theory of computation → Logic and databases; Theory of computation → Database query languages (principles); Theory of computation → Database query processing and optimization (theory); Theory of computation → Data structures design and analysis; Theory of computation → Graph algorithms analysis; Theory of computation → Streaming, sublinear and near linear time algorithms; Theory of computation → Database constraints theory; Theory of computation → Rewrite systems; Theory of computation → Regular languages; Theory of computation → Grammars and context-free languages; Theory of computation → Computational Geometry; Theory of computation → Regular languages; Theory of computation → Semantics and reasoning; Theory of computation → Automated reasoning; Theory of computation → Complexity theory and logic; Theory of computation → Data provenance

**ISBN 978-3-95977-312-6**

#### 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/978-3-95977-312-6>.

#### Publication date

March, 2024

#### Bibliographic information published by the Deutsche Nationalbibliothek

The Deutsche Nationalbibliothek lists this publication 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.ICDT.2024.0

ISBN 978-3-95977-312-6

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*

- Luca Aceto (Reykjavik University, IS and Gran Sasso Science Institute, IT)
- Christel Baier (TU Dresden, DE)
- Roberto Di Cosmo (Inria and Université de Paris, FR)
- Faith Ellen (University of Toronto, CA)
- Javier Esparza (TU München, DE)
- Daniel Král' (Masaryk University, Brno, CZ)
- Meena Mahajan (*Chair*, Institute of Mathematical Sciences, Chennai, IN)
- Anca Muscholl (University of Bordeaux, FR)
- Chih-Hao Luke Ong (University of Oxford, GB)
- Phillip Rogaway (University of California, Davis, US)
- Eva Rotenberg (Technical University of Denmark, Lyngby, DK)
- Raimund Seidel (Universität des Saarlandes, Saarbrücken, DE and Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Wadern, DE)
- Pierre Senellart (ENS, Université PSL, Paris, FR)

**ISSN 1868-8969**

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



## ■ Contents

Preface	
<i>Graham Cormode and Michael Shekelyan</i> .....	0:vii
The ICDT 2024 Test of Time Award	
.....	0:ix
Organization	
.....	0:xi
External Reviewers	
.....	0:xiii
Contributing Authors	
.....	0:xv

### Invited Talks

Natural Language Data Interfaces: A Data Access Odyssey	
<i>Georgia Koutrika</i> .....	1:1–1:22
How Database Theory Helps Teach Relational Queries in Database Education	
<i>Sudeepa Roy, Amir Gilad, Yihao Hu, Hanze Meng, Zhengjie Miao,</i> <i>Kristin Stephens-Martinez, and Jun Yang</i> .....	2:1–2:9
Rule-Based Ontologies: From Semantics to Syntax	
<i>Andreas Pieris</i> .....	3:1–3:1

### Regular Papers

Direct Access for Answers to Conjunctive Queries with Aggregation	
<i>Idan Eldar, Nofar Carmeli, and Benny Kimelfeld</i> .....	4:1–4:20
Communication Cost of Joins over Federated Data	
<i>Tamara Cucumides and Juan Reutter</i> .....	5:1–5:19
Range Entropy Queries and Partitioning	
<i>Sanjay Krishnan and Stavros Sintos</i> .....	6:1–6:21
Skyline Operators for Document Spanners	
<i>Antoine Amarilli, Benny Kimelfeld, Sébastien Labbé, and Stefan Mengel</i> .....	7:1–7:18
When Do Homomorphism Counts Help in Query Algorithms?	
<i>Balder ten Cate, Victor Dalmau, Phokion G. Kolaitis, and Wei-Lin Wu</i> .....	8:1–8:20
Approximating Single-Source Personalized PageRank with Absolute Error Guarantees	
<i>Zhewei Wei, Ji-Rong Wen, and Mingji Yang</i> .....	9:1–9:19
Right-Adjoints for Datalog Programs	
<i>Balder ten Cate, Víctor Dalmau, and Jakub Opršal</i> .....	10:1–10:20

27th International Conference on Database Theory (ICDT 2024).

Editors: Graham Cormode and Michael Shekelyan



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

On the Convergence Rate of Linear Datalog <sup>o</sup> over Stable Semirings <i>Sungjin Im, Benjamin Moseley, Hung Ngo, and Kirk Pruhs</i> .....	11:1–11:20
Enumeration and Updates for Conjunctive Linear Algebra Queries Through Expressibility <i>Thomas Muñoz Serrano, Cristian Riveros, and Stijn Vansummeren</i> .....	12:1–12:20
Direct Access for Conjunctive Queries with Negations <i>Florent Capelli and Oliver Irwin</i> .....	13:1–13:20
The Importance of Parameters in Database Queries <i>Martin Grohe, Benny Kimelfeld, Peter Lindner, and Christoph Standke</i> .....	14:1–14:17
Conjunctive Queries on Probabilistic Graphs: The Limits of Approximability <i>Antoine Amarilli, Timothy van Bremen, and Kuldeep S. Meel</i> .....	15:1–15:20
Optimally Rewriting Formulas and Database Queries: A Confluence of Term Rewriting, Structural Decomposition, and Complexity <i>Hubie Chen and Stefan Mengel</i> .....	16:1–16:17
Containment of Regular Path Queries Under Path Constraints <i>Sylvain Salvati and Sophie Tison</i> .....	17:1–17:19
Computing Data Distribution from Query Selectivities <i>Pankaj K. Agarwal, Rahul Raychaudhury, Stavros Sintos, and Jun Yang</i> .....	18:1–18:20
Information Inequality Problem over Set Functions <i>Miika Hannula</i> .....	19:1–19:20
Conditional Independence on Semiring Relations <i>Miika Hannula</i> .....	20:1–20:20
Subgraph Enumeration in Optimal I/O Complexity <i>Shiyuan Deng and Yufei Tao</i> .....	21:1–21:20
Evaluating Graph Queries Using Semantic Treewidth <i>Cristina Feier, Tomasz Gogacz, and Filip Murlak</i> .....	22:1–22:20
Join Sampling Under Acyclic Degree Constraints and (Cyclic) Subgraph Sampling <i>Ru Wang and Yufei Tao</i> .....	23:1–23:20
Finding Smallest Witnesses for Conjunctive Queries <i>Xiao Hu and Stavros Sintos</i> .....	24:1–24:20
Ranked Enumeration for MSO on Trees via Knowledge Compilation <i>Antoine Amarilli, Pierre Bourhis, Florent Capelli, and Mikaël Monet</i> .....	25:1–25:18

## ■ Preface

The 27th International Conference on Database Theory (ICDT 2024) was held in Paestum, Italy, from March 25th to March 28th, 2024. The Program Committee has selected 22 research papers out of 74 submissions for publication at the conference.

The PC has further decided to give the best paper award to:

*Finding Smallest Witnesses for Conjunctive Queries*  
**Xiao Hu and Stavros Sintos**

The ICDT 2024 best newcomer award goes to:

*Direct Access for Conjunctive Queries with Negation*  
**Florent Capelli and Oliver Irwin**

We congratulate the winners!

Apart from the 22 regular papers, these proceedings include papers accompanying the invited (shared) EDBT/ICDT keynotes by Sudeepa Roy (Duke University) and the ICDT invited talk by Andreas Pieris (University of Edinburgh and University of Cyprus).

A committee formed by Nofar Carmeli, Reinhard Pichler and Nicole Schweikardt has decided to give the Test of Time Award for ICDT 2024 to the ICDT 2014 paper:

*Leapfrog Triejoin: A Simple, Worst-Case Optimal Join Algorithm*  
**Todd L. Veldhuizen**

We would like to thank all people who contributed to the success of ICDT 2024, including the authors of all submitted papers, keynote and invited talk speakers, and, of course, all members of the Program Committee as well as the external reviewers, for the very substantial work that they have invested over the two submission cycles of ICDT 2024. Their effort and wisdom were critical to ensure that the final program of the conference satisfies the highest standards. We would also like to thank the ICDT Council members for their support on a wide variety of matters; the program chairs of EDBT, Letizia Tanca and Qiong Luo, for their assistance; and the local organizers of the EDBT/ICDT 2024 conference, led by General Chairs Giuseppe Polese and Loredana Caruccio, for the great job they did in organizing the conference and co-located events.

Finally, we wish to acknowledge Dagstuhl Publishing for their support with the publication of the proceedings in the LIPIcs (Leibniz International Proceedings in Informatics) series.

Graham Cormode and Michael Shekelyan  
March 2024







## ■ The ICDT 2024 Test of Time Award

In 2013, the International Conference on Database Theory (ICDT) began awarding the ICDT Test-of-Time (ToT) award, with the goal of recognizing one paper, or a small number of papers, presented at earlier ICDT conferences that have best met the “test of time”. In 2024, the award recognizes a paper selected from the proceedings of the ICDT 2014 conference that has had the highest impact in terms of research, methodology, conceptual contribution, or transfer to practice over the past decade. The award was presented during the EDBT/ICDT 2024 Joint Conference, March 25–28, 2024 in Paestum, Italy.

The 2024 ToT Award Committee consists of Nofar Carmeli, Reinhard Pichler, and Nicole Schweikardt (chair). After careful consideration and soliciting external assessments, the committee has chosen the following contribution for the 2024 ICDT Test-of-Time Award:

*Leapfrog Triejoin: A Simple, Worst-Case Optimal Join Algorithm*  
**Todd L. Veldhuizen**

This paper introduced the worst-case optimal join algorithm called *Leapfrog Triejoin* (LFTJ). What singles out LFTJ from previously published worst-case optimal join algorithms is that it is very intuitive, simple to describe and easy to implement. It is based on backtracking search, and in contrast to many previously used join algorithms it follows the variable-at-a-time paradigm rather than the classical relation-at-a-time paradigm. Variants of LFTJ have found their way into open-source and commercial database systems, including LogicBlox, Umbra, Kùzu, RelationalAI, FreeJoin, EmptyHeaded, FBENCH, etc.

Apart from its impact on systems, the paper also contains an entirely new technique for proving LFTJ’s worst-case optimality. While earlier proofs for other worst-case optimal join algorithms used a sophisticated entropy-based machinery, Todd L. Veldhuizen presents a self-contained and intuitively simple proof technique. This proof shows that for any database instance on which LFTJ runs in  $n$  steps, there exists another database instance with the same relation sizes, where the query answer has size  $n$ . This implies that the running time is bounded by the query size on some instance, and therefore it is bounded by the general query size upper bound. Due to its optimality, elegance, and simplicity, the algorithm as well as the proof method today are taught in university courses on the principles of database systems.

In the last decade, LFTJ has greatly influenced theoretical as well as practical database research. For all these reasons, Todd L. Veldhuizen’s ICDT 2014 paper is one of the great gems of ICDT.

Nofar Carmeli  
Inria and Univ. Montpellier

Reinhard Pichler  
TU Wien

Nicole Schweikardt  
HU Berlin

*The ICDT Test-of-Time Award Committee for 2024*





## ■ Organization

### General Chairs

Giuseppe Polese, University of Salerno

Loredana Caruccio, University of Salerno

Markus Schmid, Humboldt-Universität

Yufei Tao, CUHK

Jef Wijsen, University of Mons

### Program Chair

Graham Cormode, University of Warwick

### Proceedings Chair

Michael Shekelyan, Queen Mary University of London

### Program Committee

Mahmoud Abo Khamis, Relational.AI

Antoine Amarilli, Telecom Paris

Yael Amsterdamer, Bar Ilan University

Pablo Barceló, PUC Chile

Vladimir Braverman, Rice University

Marco Calautti, Università degli Studi di Milano

Hubie Chen, King's College London

Cristina Feier, University of Warsaw

Diego Figueira, Université de Bordeaux

Dominik Freydenberger, Loughborough University

Martin Grohe, RWTH Aachen University

Daniel Kifer, Penn State University

Benny Kimelfeld, Technion

Paraschos Koutris, University of Wisconsin-Madison

Stefan Mengel, CNRS

Frank Neven, Hasselt University

Matthias Niewerth, Bayreuth University

Jeff M. Phillips, University of Utah

Reinhard Pichler, TU Wien

Cristian Riveros, PUC Chile

Francesco Scarcello, Università della Calabria

27th International Conference on Database Theory (ICDT 2024).

Editors: Graham Cormode and Michael Shekelyan



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



## ■ External Reviewers

Artur Czumaj

Cibele Freire

Cinzia Marté

Davide Mario Longo

Florent Capelli

Hangdong Zhao

Harry Vinnal-Smeeth

Jelle Hellings

Ke Yi

Marcelo Arenas

Nofar Carmeli

Peyman Afshani

Rana Shahout

Rémi Morvan

Sam Thompson

Samson Zhou

Sandeep Silwal

Sudeepa Roy

Xuan Wu

Yuxin Tang




## ■ Contributing Authors

Pankaj K. Agarwal  (18)

Department of Computer Science, Duke University,  
Durham, NC, USA

Antoine Amarilli  (7, 15, 25)

LTCI, Télécom Paris, Institut Polytechnique de Paris, France

Pierre Bourhis  (25)

Univ. Lille, CNRS, Inria, Centrale Lille, UMR 9189 CRISTAL, F-59000 Lille, France

Florent Capelli  (13, 25)

Univ. Artois, CNRS, UMR 8188, Centre de Recherche en Informatique de Lens (CRIL), F-62300 Lens, France

Nofar Carmeli  (4)

Inria, LIRMM, Univ Montpellier, CNRS, France

Hubie Chen (16)

Department of Informatics, King's College London, UK

Tamara Cucumides (5)

University of Antwerp, Belgium

Victor Dalmau  (8)

Universitat Pompeu Fabra, Barcelona, Spain

Víctor Dalmau  (10)

Department of Information and Communication Technologies, Universitat Pompeu Fabra, Barcelona, Spain

Shiyuan Deng (21)


The Chinese University of Hong Kong, China

Idan Eldar  (4)

Technion – Israel Institute of Technology, Haifa, Israel

Cristina Feier (22)

University of Warsaw, Poland

Amir Gilad  (2)


Hebrew University of Jerusalem, Israel

Tomasz Gogacz (22)


University of Warsaw, Poland

Martin Grohe  (14)

RWTH Aachen University, Germany

Miika Hannula  (19, 20)

University of Helsinki, Finland

Xiao Hu  (24)


University of Waterloo, Canada

Yihao Hu  (2)

Duke University, Durham, NC, USA

Sungjin Im (11)

University of California, Merced, CA, USA

Oliver Irwin  (13)

Université de Lille, CNRS, Inria, UMR 9189 – CRISTAL, F-59000 Lille, France

Benny Kimelfeld  (4, 7, 14)

Technion – Israel Institute of Technology, Haifa, Israel

Phokion G. Kolaitis  (8)

University of California Santa Cruz, CA, USA; IBM Almaden Research Center, San Jose, CA, USA

Georgia Koutrika  (1)


Athena Research Center, Athens, Greece

Sanjay Krishnan  (6)


Department of Computer Science, University of Chicago, IL, USA

Sébastien Labbé (7)

École normale supérieure, Paris, France

Peter Lindner  (14)

École Polytechnique Fédérale de Lausanne, Switzerland

Kuldeep S. Meel  (15)

University of Toronto, Canada

Hanze Meng  (2)

Duke University, Durham, NC, USA

Stefan Mengel (7, 16)

Univ. Artois, CNRS, Centre de Recherche en Informatique de Lens (CRIL), France

Zhengjie Miao  (2)

Simon Fraser University, Burnaby, Canada

Mikaël Monet  (25)

Université de Lille, CNRS, Inria, UMR 9189 – CRISTAL, F-59000 Lille, France

Benjamin Moseley (11)

Carnegie Mellon University, Pittsburgh, PA, USA

Filip Murlak (22)

University of Warsaw, Poland


27th International Conference on Database Theory (ICDT 2024).

Editors: Graham Cormode and Michael Shekelyan





Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Thomas Muñoz Serrano  (12)  
UHasselt, Data Science Institute,  
Diepenbeek, Belgium

Hung Ngo (11)  
RelationalAI, Berkeley, CA, USA


Jakub Opršal  (10)  
School of Computer Science,  
University of Birmingham, UK


Andreas Pieris  (3)  
University of Edinburgh, UK;  
University of Cyprus, Nicosia, Cyprus


Kirk Pruhs (11)  
University of Pittsburgh, Pittsburgh, PA, USA


Rahul Raychaudhury (18)  
Department of Computer Science, Duke  
University, Durham, NC, USA


Juan Reutter (5)  
PUC Chile & IMFD Chile, Santiago, Chile

Cristian Riveros  (12)  
Pontificia Universidad Católica de Chile,  
Santiago, Chile; Millennium Institute for  
Foundational Research on Data, Santiago, Chile

Sudeepa Roy  (2)  
Duke University, Durham, NC, USA;  
RelationalAI, Berkeley, CA, USA  
(Visiting Scientist)

Sylvain Salvati  (17)  
Université de Lille, INRIA, CRISTAL/CNRS  
UMR 9189, France


Stavros Sintos  (6, 18, 24)  
Department of Computer Science,  
University of Illinois at Chicago, IL, USA


Christoph Standke  (14)  
RWTH Aachen University, Germany

Kristin Stephens-Martinez  (2)  
Duke University, Durham NC, USA

Yufei Tao (21, 23)  
The Chinese University of Hong Kong, China


Balder ten Cate  (8, 10)  
University of Amsterdam, The Netherlands


Sophie Tison  (17)  
Université de Lille, INRIA, CRISTAL/CNRS  
UMR 9189, France


Timothy van Bremen  (15)  
National University of Singapore, Singapore


Stijn Vansummeren  (12)  
UHasselt, Data Science Institute,  
Diepenbeek, Belgium


Ru Wang (23)  
The Chinese University of Hong Kong, China

Zhewei Wei  (9)  
Renmin University of China, Beijing, China

Ji-Rong Wen  (9)  
Renmin University of China, Beijing, China


Wei-Lin Wu  (8)  
University of California Santa Cruz, CA, USA

Jun Yang  (2, 18)  
Duke University, Durham, NC, USA

Mingji Yang  (9)  
Renmin University of China, Beijing, China



# Natural Language Data Interfaces: A Data Access Odyssey

Georgia Koutrika   

Athena Research Center, Athens, Greece

---

## Abstract

Back in 1970's, E. F. Codd worked on a prototype of a natural language question and answer application that would sit on top of a relational database system. Soon, natural language interfaces for databases (NLIDBs) became the holy grail for the database community. Different approaches have been proposed from the database, machine learning and NLP communities. Interest in the topic has had its peaks and valleys. After a long and adventurous journey of almost 50 years, there is a rekindled interest in NLIDBs in recent years, fueled by the need for democratizing data access and by the recent advances in deep learning and natural language processing in particular. There is a surge of works on natural language interfaces for databases using neural translation, and suddenly it becomes hard to keep up with advancements in the field. Are we close to finding the holy grail of data access? What are the lurking challenges that we need to surpass and what research opportunities arise? Finally, what is the role of the database community?

**2012 ACM Subject Classification** Computing methodologies → Machine translation; Information systems → Data management systems

**Keywords and phrases** natural language data interfaces, NLIDBs, NL-to-SQL, text-to-SQL, conversational databases

**Digital Object Identifier** 10.4230/LIPIcs.ICDT.2024.1

**Category** Invited Talk

## 1 Introduction

E. F. Codd, the father of relational databases, said that “*If we are to satisfy the needs of casual users of databases, we must break the barriers that presently prevent these users from freely employing their native language*” [11]. Throughout the 1970s, he worked on a prototype of a natural language question and answer application that would sit on top of a relational database system, called Rendezvous. Rendezvous allowed a user with no knowledge of database systems – and even limited knowledge of a given database’s content – to engage in a dialog with the system. Almost 50 years after, natural language interfaces for databases (NLIDBs) are into the spotlight.

NLIDBs or *natural language data interfaces* are appealing for a number of reasons [3]. They are more suitable for occasional users, alleviating the need for the user to spend time learning the system’s query language to access data. Some questions are easier expressed in natural language (e.g., questions involving negation, or quantification). Moreover, natural language (NL) questions can be brief and support anaphoric and elliptical expressions, where the meaning of each question is complemented by the discourse context.

The first NLIDBs appeared back in the sixties. For instance, LADDER was developed as a management aid to Navy decision makers [31]. These early systems interfaced application-specific non-SQL database systems, and they could not be used with different data. Several approaches have followed over the years focusing on translating NL questions to SQL over relational data. Industrial systems also made their appearance. For example, in the late 90’s, Microsoft SQL Server shipped with the English Query feature. Some systems enabled keyword searches. They relied on data indexes to find relations that contained the query



© Georgia Koutrika;  
licensed under Creative Commons License CC-BY 4.0  
27th International Conference on Database Theory (ICDT 2024).

Editors: Graham Cormode and Michael Shekelyan; Article No. 1; pp. 1:1–1:22



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

keywords and on the database schema to join them and return the answer to a query (e.g., [32, 51]). Parsing-based approaches parsed the input question to understand its grammatical structure and then map it to the structure of the desired SQL query (e.g., [47, 77]).

Recently, the use of deep learning techniques, and in particular LLMs, has given a great boost in the development of NLIDBs [22, 27, 38, 48, 56, 86]. The creation of two large datasets, WikiSQL [86] and Spider [81], for training NLIDBs has brought several developments in this field, with new systems popping up like mushrooms. These have popularized the term *Text-to-SQL* (or NL-to-SQL) to refer to NLIDBs that focus on translating NL questions to SQL. For the first time, it seems possible that technological barriers can be broken and human-like interaction with data can become a reality.

## 2 The Inherent Challenges of Natural Language

While using natural language as a query language appears very appealing, it also brings challenges in query understanding and answering that standard database query languages, such as SQL, are free of, by design. These challenges, however, plague NLIDBs.

**Ambiguity.** Natural language is ambiguous for human-computer interaction. Ambiguity allows more than one interpretation. Unfortunately, there are several types of ambiguity that a NLIDB needs to resolve [25]. For example, “*Paris*” may refer to the city or a person (lexical ambiguity). The question “*Find all German movie directors*” can be parsed into “*directors that have directed German movies*” or “*directors from Germany that have directed a movie*” (syntactic ambiguity). The question “*Are Brad and Angelina married?*” is an example of semantic ambiguity, as it is unsure if it means they are married to each other or separately. Context-dependent ambiguity refers to a term having different meanings in different contexts. For example, “*top*” in “*top scorer*” means the highest (total) number of goals, while in “*top movies*”, it signifies the greatest rating. As a result of ambiguity, there may be multiple potential interpretations of a natural language query. Generating and processing them takes a toll on the system efficiency. It also affects the system’s effectiveness, i.e., its ability to find which interpretation captures the user intent and correctly answer the user query.

**Paraphrasing.** Completely different words or sentences can have the same meaning. For instance, “*How many people live in Amsterdam?*” and “*What is the population of Amsterdam?*”. Dealing with different NL utterances (paraphrasing) is a challenge, as each one may need different handling. For instance, the second NL query may be easier for a system because it is likely that a *population* attribute exists in the database schema.

**Inference.** A natural language sentence may not contain all information needed for a system to fully understand it. In *elliptical queries*, one or more words are omitted but can still be understood in the context of the sentence. An example is “*Who was the president before Obama*”. The fact that the query refers to US presidents needs to be inferred. In *follow-up questions*, which are common in conversations between humans, missing information can be understood in the context of the dialog. We ask a question, receive an answer, and then ask a follow-up question assuming that the context of the first question is known. For example, “*Q: Which is the capital of Germany?*”, “*A: Berlin*”, “*Q: What about France?*”. The system has to infer that the user is asking for the capital of France.

**User mistakes.** Spelling, syntactical or grammatical errors can be understood by a human but are hard to be recognized and ignored by a system. For example, if the user asks for “*movies by Brad Bitt*”, can the system recognize that the user refers to Brad Pitt, and make the necessary correction, and not to a Brad Bitt that the system does not happen to know based on the underlying data?

### 3 A trip down Memory Lane

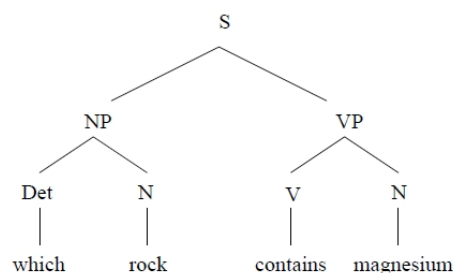
Prototype NLIDBs appeared in the late sixties and early seventies [12, 63]. For example, Lunar [74] was a natural language interface to a database with chemical analyses of moon rocks. LADDER [31] used semantic grammars, a technique that interleaves syntactic and semantic processing. Early eighties witnessed a lot of research on NLIDBS [3]. For instance, Chat-80 was implemented entirely in Prolog [72]. It transformed English questions into Prolog expressions, which were evaluated against a Prolog database. A number of commercial systems, such as IBM’s LanguageAccess [53] and Intellect [28] from Trinzic, appeared.

Some of the early systems relied on *pattern-matching* techniques to answer the user’s questions (e.g., [36]). A primitive pattern-matching system could use rules like:

```
pattern: ... “capital” ... <country>
action : Report Capital of row where Country = <country>
```

This rule says that if a user’s request contains the word “*capital*” followed by a country name (i.e., a name appearing in the Country column), then the system should locate the row which contains the country name, and print the corresponding capital.

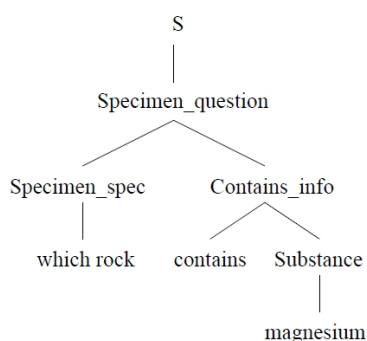
*Syntax-based systems* used a grammar that described the possible syntactic structures of the user’s questions (e.g., Lunar [74]). A user question would be first parsed (i.e. analysed syntactically) resulting in a syntax tree like the one shown in Figure 1. Then, the resulting parse tree would be directly mapped to a query in some database query language. To perform this mapping, the system would use specific mapping rules that would specify how each part of the tree would map to a part of the database query. These syntax-based NLIDBs usually interfaced to application-specific database systems that provided database query languages carefully designed to facilitate the mapping from the parse tree to the database query. At this point, it was usually difficult to devise mapping rules to transform directly the parse tree into some expression in a real-life database query language (e.g. SQL) [3].



■ **Figure 1** An example syntax tree from Lunar [3].

In *semantic-grammar systems* (e.g., LADDER [31], EUFID [63]), the question-answering is still done by parsing the input and mapping the parse tree to a database query. The difference is that the grammar categories (i.e., the non-leaf nodes that will appear in the parse tree) would correspond to semantic concepts (e.g., Substance, Radiation, or Specimen) instead of syntactic constituents (e.g., noun-phrase, noun, sentence). An example tree is

shown in Figure 2. Semantic grammars contain hard-wired knowledge about a specific knowledge domain, and semantic grammar categories are usually chosen to enforce semantic constraints. Since the grammar is domain-specific, such systems are very difficult to port to other knowledge domains.



■ **Figure 2** An example semantic tree [3].

Most recent systems in that period would first transform the natural language question into an *intermediate logical query*, that expresses the meaning of the user question in terms of high-level world concepts, which are independent of the database structure. The logical query is then translated to an expression in the database query language (e.g., CLE [2], TEAM [26]). For example, the Essex system [15] used a principled multi-stage transformation process: the system first generated a logic query, expressed in a version of untyped  $\lambda$ -calculus, which was then transformed into a first-order predicate logic expression, which was subsequently translated into universal-domain relational calculus, domain relational calculus, tuple relational calculus, and finally SQL.

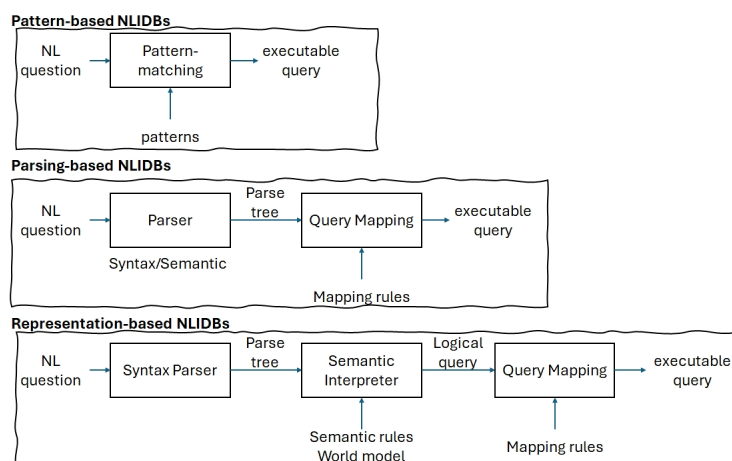
In many systems, the syntax rules linking non-terminal symbols (non-leaf nodes in the parse tree) and the corresponding semantic rules are domain-independent; i.e., they could be used in any application domain. The information, however, describing the possible words (leaf nodes) and the logic expressions corresponding to the possible words is domain-dependent, and has to be declared in the lexicon. The logic query does not refer to database objects (e.g. tables, columns), and it does not specify how to search the database to retrieve the necessary information. In order to retrieve the information requested by the user, the logic query has to be transformed into a query expressed in some database query language supported by the underlying DBMS, using the mapping to database information.

Figure 3 captures the evolution of systems during this period. In early NLIDBs, the syntax/semantics rules were based on rather ad hoc ideas, and expressed in idiosyncratic formalisms. In the nineties and later, systems follow some representation-based approach and adopt advances in the general natural language processing field, for better syntactic and semantic parsing and interpretation.

## 4 The Database-Way Era

After 2000, the next 15 years witness the emergence of systems that focus on translating keyword or NL queries to SQL. They adopt the general architecture shown in Figure 4 [25].

The *parser* is responsible for gathering linguistic information on the user query. Its output varies in complexity ranging from a simple set containing meaningful keywords (e.g., *movie*, *actor*, “*Brad Pitt*”) (e.g., Discover [33]), to a fully structured parse tree (e.g., NaLIR[47]).



■ **Figure 3** Early NLIDBs system architectures.

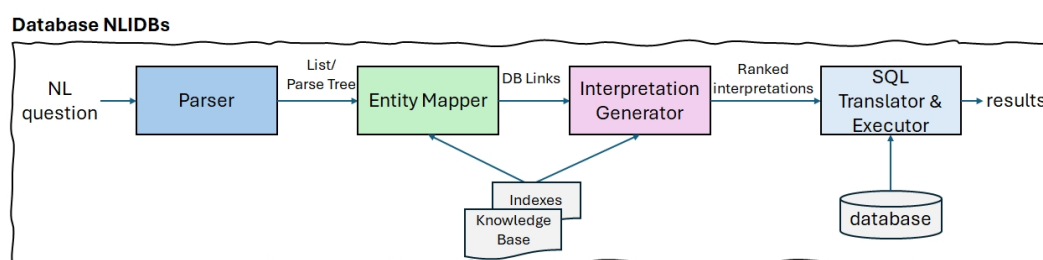
The *entity mapper* is responsible for mapping the terms extracted by the Parser to database elements. Some systems use inverted indexes to map the query terms to values in the database (e.g., Discover [33]). Others also search the database metadata, such as relation and attribute names (e.g., DiscoverIR[32]). NaLIR [47] also identifies the nodes in the parse tree that can be mapped to SQL elements (select, operator, function, quantifier and logic nodes) using a knowledge base of phrases – for example, the function node *COUNT* corresponds to the phrase “number of”.

The *interpretation generator* infers the semantics of the query as a whole. Dependencies between terms and phrases are analyzed in order to extract joins, aggregate functions, comparisons, etc., depending on the operations that every NLIDB system supports. The output is an intermediate well-defined structured format that captures the meaning of the query and the order of the operations to be done. This intermediate representation is an *interpretation* of the query from the system’s perspective. Intermediate representations are useful because they are easier to modify, manipulate, and rank, as opposed to SQL queries. Ambiguity at the term level can be transferred from the output of the entity mapper, due to multiple mappings for a term. Ambiguity also exists in the linguistic dependencies between terms, in the extraction of joins, in the order of the operations, and so forth. As a result, the output of this step may be *multiple candidate interpretations* ranked according to how well the system thinks an interpretation captures the user intent.

*Indexes and knowledge bases* are used by the entity mapper and interpretation generator. The database inverted indexes are used to map input terms extracted by the parser to database values. Knowledge bases are additional sources that a system can exploit to understand the query, such as dictionaries containing word definitions and context-specific definitions, synonym lists, grammar rules and syntactic patterns.

The *SQL translator & executor* translates the intermediate interpretation to SQL. This task heavily depends on the quality of the highest ranked interpretations, since only a selected few are executed. Each system has its own set of rules to convert an intermediate representation into a syntactically correct SQL query. Some systems may rank the results returned by every query, which means that the tuples returned by one query do not always adopt the score of that query that produced them, but may have their own ranking.

These approaches bring the following notable novelties (a detailed experimental evaluation of representative systems sheds light into their capabilities [25]):

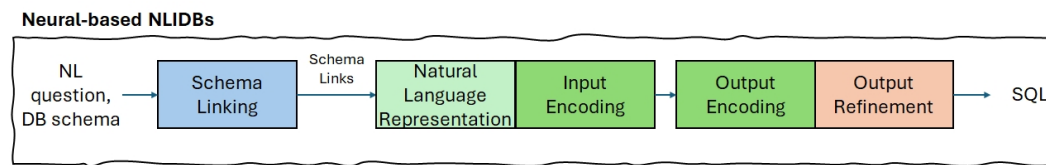


■ **Figure 4** Database NLIDBs system architecture.

**(1) A database mapping problem.** Query translation is viewed as a database mapping problem: mapping query terms to database elements (tables, columns and values) and finding the desired interconnections of these data elements that capture the user intent. For this reason, the entity mapper, which maps the query terms to database elements, comes right after the parser as a crucial step towards understanding the query from the perspective of the database. Together, they ensure that the final SQL will be *semantically correct*. Note that, in earlier, representation-based, systems, this mapping would happen at a later stage through the semantic interpreter (using rules and some model of the world). Discover [33] introduced the concept of *generating query interpretations as subgraphs of the database schema graph*, called *joining networks* or (trees), adopted by several systems (e.g., DiscoverIR [32], Spark [51], SODA [4]). The interpretation generator constructs multiple joining networks that connect the relations that contain the query terms through edges that are the primary key-foreign key relationships between them. Précis [41, 62] extended the concept of joining networks and introduced the *logical database subset* that contains not only items directly related to the given query keywords but also items implicitly related to them, with the purpose of providing to the user greater insight into the data. ATHENA [57] translated the input natural language query (NLQ) into an intermediate query language over a domain ontology, which was subsequently translated into SQL.

**(2) Query disambiguation.** Dealing with the ambiguity of natural language is important for these systems. Query disambiguation is performed at two levels: interpreting a single term (entity mapper), and interpreting the whole query (interpretation generator). A query term may have *multiple* potential mappings to database elements. The entity mapper may assign a score to each possible mapping – for example, an information retrieval-style score (e.g., [32]) – in order to distinguish between likely and not likely mappings. On the other hand, terms may not map directly to any database element due to synonyms and non-exact matches. The system may attempt to find the closest mappings using a knowledge base with synonyms (e.g., SODA [4]), or a string similarity algorithm (e.g., NaLIR[47]). To minimize the chances of ambiguity and properly interpret the query intent, some systems impose stricter syntactic constraints on their input (e.g., [4], [83]). The position of the keywords, functions and operators matters and minor changes can alter the semantics or even render the query incomprehensible. For example, a query could be “*count movies actor “Brad Pitt”*”. Using this approach, ExpressQ [83] was the first system to accept keyword queries with aggregate functions and `groupBy`. The interpretation generator may rank query interpretations using simple ranking schemes, like the size of the joining network (Discover [33]), the sum of term scores normalized by the tree size ([32]) or a function of the edge weights [62]. More elaborated formulas have been used. For example, Spark [51] models a joining tree as a *virtual document*, and computes its score as a function of an information retrieval score, a





■ **Figure 5** Neural NLIDBs system architecture.

completeness factor that quantifies how many different query keywords are included in the tree, and the tree size. NaLIR [47] ranks query interpretations based on: (i) the number of parse tree nodes that violate the grammar, (ii) the mappings between the parse tree nodes and the database elements that can help infer the desired structure of a parse tree, and (iii) the similarity to the original linguistic parse tree.

**(3) Execution-coupled translation.** A characteristic of these systems is that they are responsible not just for translating the query but also for generating the final response to the query. The SQL translator & executor ensures the *syntactic correctness of the generated SQL queries*. The response is ranked according to how well it matches the user query. A lot of effort has been put into the execution algorithm, aiming to return the top-k results faster. To achieve this, heuristics are applied to stop the execution without materializing all the possible joining trees (e.g., DiscoverIR [32]), or accessing fewer parts of the data (Spark [51]). Précis [41] progressively populates the relations in a logical subset to reduce the number of database rows fetched to the ones needed in the answer. ▽

Early systems of this period are keyword-based, adopting the fundamental characteristic of search engines [1]. NaLIR [47] supported NL queries and used the Stanford Parser [14] to generate a dependency parse tree, where nodes represent the terms and edges represent the linguistic relationships between them. Its interpretation generator corrected any possible flaws in the structure of the parse tree to make it grammatically valid. ATHENA++ [59] combines linguistic patterns from NL queries with deep domain reasoning using ontologies to enable nested query detection and generation for specific domains.

## 5 The Deep Learning Era

If in the end of the nineties, “*the development of NLIDBs is no longer as fashionable a topic within academic research*” [3], after 2017, there has been an explosion of works on NLIDBs [38, 37]. These approaches tackle the text-to-SQL problem as a *language translation problem*, and train a neural network on a large amount of {NL query/SQL} pairs [38].

Figure 5 shows the architecture of a neural NLIDB [38]. In its core lies the neural network that typically follows an encoder-decoder architecture. The encoder takes one or more inputs of variable shapes and transforms them into one or more internal representations with fixed shapes that are consumed by the decoder. Additionally, the encoder usually infuses the representation of each input with information from the rest of the inputs, so as to create a more informed representation that better captures the instance of the problem at hand, through a mechanism called neural attention. The decoder uses the representations calculated by the encoder and makes predictions on the most probable SQL query (or parts of it).

Given that the inputs of an NLIDB are mainly textual, the *natural language representation* creates an efficient numerical representation of the input that can be accepted by the encoder. Early systems used pre-trained word embeddings for NL representation, such as GloVe [54] (e.g., SQLNet [75], IncSQL [61]). Recent systems rely on Pre-trained Language Models (PLMs) such as BERT [16] (e.g., HydraNet [52], ValueNet [7]), that provide better representations.

Apart from the NL question, the inputs to an NLIDB may include table and column names, values, primary-to-foreign key relationships and relationships between columns and tables. *Input encoding* structures the inputs so that the encoder can process them. Options range from encoding each column with the NL query separately as in HydraNet [52] to schema graph encoding (e.g., RAT-SQL [68]). *Output decoding* consists of designing the structure of the predictions that the network will make, as well as choosing the appropriate network for making such predictions (e.g., a SQL query can be viewed as a simple string, or as a structured program which follows a certain grammar).

Neural NLIDBs have the following notable features (see [38] for a detailed survey):

**(1) A neural translation problem.** Viewing the problem as a language translation one, neural text-to-SQL systems take as input a NL query and return a SQL query. Depending on how their decoder generates the output, they can be divided into three categories [10]: (a) sequence-based, (b) sketch-based slot-filling, and (c) grammar-based approaches.

*Sequence-based approaches* generate the predicted SQL, or a large part of it, as a sequence of words (comprising SQL tokens and schema elements) [8, 50, 86]. Seq2SQL [86] was one of the first neural networks created specifically for the text-to-SQL task and was based on a previous work focusing on generating logical forms using neural networks [17]. The system predicts an aggregation function and the column for the SELECT clause as classification tasks and generates the WHERE condition clause using a seq-to-seq network. The latter network is burdened with generating parts of the query and can lead to syntactic errors, which is its major drawback. The network architecture combines LSTM and linear layers, and its inputs are represented as GloVe embeddings. More recent sequence-based approaches are more effective thanks to the use of large pre-trained seq-to-seq Transformer [67] models (e.g., T5 [55], BART [46]) and the use of smarter decoding techniques that constrain the predictions of the decoder and prevent it from producing invalid queries (e.g., PICARD [58]).

*Sketch-based slot-filling approaches* (e.g., XSQL [30], SLOVA [35], HydraNet [52], SQLNet [75]) consider a query sketch with a number of empty slots that must be filled in, and use neural networks to predict the most probable elements for each slot, such as the table columns that appear in the SELECT clause. In this way, the SQL generation task is transformed into a set of classification tasks. SQLNet [75], one of the first sketch-based approaches, was based on the observation that the way Seq2SQL [86] chose to generate the WHERE clause was prone to errors that could be avoided. For this reason, a set of query sketches were developed and separate neural networks were created to fill each type of slot. While dividing the text-to-SQL problem into small sub-tasks makes it easier to generate syntactically correct queries, sketch-based approaches have two drawbacks. Firstly, the resulting neural network architecture may end up being quite complex since dedicated networks may be used for each slot or part of the query. Furthermore, it is hard to extend to complex SQL queries, because generating sketches for any type of SQL query is not trivial.

*Grammar-based approaches* (e.g., RyanSQL [10], IRNet [27], IncSQL [61], Rat-SQL [68]) produce a sequence of grammar rules instead of simple tokens in their output. These grammar rules are instructions that, when applied, can create a structured query. The most often used grammar-based decoders by text-to-SQL systems have been previously proposed for code generation as an Abstract Syntax Tree (AST) [78, 79]. These models take into account the grammar of the target code language (in our case, the SQL grammar) and consider the target program to be an AST, whose nodes are expanded at every tree level using the grammar rules, until all branches reach a terminal rule. When it reaches a terminal rule, the model might generate a token, for example, a table name, an operator or a condition value, in the



case of text-to-SQL. The decoder uses a LSTM-based architecture that predicts a sequence of actions, where each action is the next rule to apply to the program AST. Because the available predictions are based both on the given grammar and the current state of the AST, the possibility of generating a grammatically incorrect query is greatly reduced.

**(2) Learning schema linking.** Schema linking aims at the discovery of possible mentions of database elements in the NL question. For this purpose, query candidates are extracted from the question and are matched to database candidates from the underlying database. Query candidates may be single words, n-grams (IRNet [27]), or named entities (ValueNet [7], TypeSQL [80]). Database candidates are tables, columns, and values stored in the database. To accelerate the search of discovered query candidates in the database values, indexes have been widely used in earlier, non-neural, text-to-SQL systems [32, 47]. ValueNet [7] also adopts this approach. The second part of schema linking is the process of mapping the query candidates to database candidates. Each mapping is called a *schema link*. These discovered schema links are fed into the neural network that is responsible for the translation.

The mapping can be performed using exact or partial matching (IRNet [27]) and approximate string matching (ValueNet [7]). Text-to-SQL systems [5, 6] have also used learned word embeddings from the area of semantic parsing [42]. The system learns word embeddings using the words of the text-to-SQL training corpus and combines them with additional features that are calculated using NER, edit distance and indicators for exact token and lemma match. These embeddings are then used to calculate the similarity of query candidates to DB candidates. While this approach is expensive, it allows for more flexible and intelligent matching. Given the complexity of schema linking, it is also possible to train a model to perform schema linking with better results. For example, a Conditional Random Field (CRF) model [43] can be trained on a small group of hand-labelled samples to recognize column links, table links and value links for numerical and textual values [8].

SDSQL [34] follows a very different approach. It is simultaneously trained on two tasks: (a) the text-to-SQL task, similarly to all systems, and (b) the *Schema Dependency Learning* task for discovering schema links using a deep biaffine network [18, 19]. In this case, the schema links discovered by the system are not directly used for predicting the SQL query; still, training for both tasks simultaneously has a positive effect on the system performance.

*Neural Attention.* While attention layers do not directly determine a match, they can highlight connections between query and DB candidates, which can improve the system’s internal representation and boost its performance. SQLNet [75] was the first system to introduce such a mechanism, named *Column Attention*, that processes the NLQ and column names and finds relevant columns for each word of the NLQ. PLMs based on the Transformer neural architecture [67], which encapsulates an attention mechanism, have become very popular for input encoding, greatly benefiting the accuracy of text-to-SQL systems (e.g., [58]). RAT-SQL [68] proposed a modified Transformer layer, called Relation-Aware Transformer (RAT), that biases the attention mechanism of the Transformer towards already-known relations from the DB schema and discovered schema links.

**(3) Output refinement.** Output refinement can be applied on a trained model to avoid producing incorrect SQL queries. *Execution-guided decoding* [70] can execute partially complete SQL queries at prediction time and decide to avoid a certain prediction if the execution fails or if it returns an empty output. Execution-guided decoding is system-agnostic and can increase the system accuracy. *Constrained decoding* is a method for incrementally parsing and constraining auto-regressive decoders, to prevent them from

producing grammatical or syntactical errors (PICARD [58]). For each token prediction, PICARD examines the generated sequence so far along with the  $k$  most probable next tokens and discards all tokens that would produce a grammatically incorrect SQL query, use an attribute that is not present in the DB at hand, or use a table column without having its table in the query scope. Other systems with sequence-based decoders have proposed similar decoding techniques to avoid errors (e.g., SeaD [76] and BRIDGE [50]).

**(4) Datasets & evaluation.** A *text-to-SQL dataset* (or *benchmark*) is a set of NL/SQL query pairs defined over one or more databases, used to train and evaluate neural text-to-SQL systems. Text-to-SQL datasets become a critical asset due to their integral role in enabling the development of such systems and serving as a common reference for evaluation. Previous, non-neural systems did not use common datasets, instead they employed a variety of small datasets that combined different databases and query sets of varying size and complexity. The lack of a common dataset to be used by different system evaluations impeded a fair system comparison and a clear view of the text-to-SQL landscape in previous years. This situation drastically changes with the emergence of WikiSQL [86] and Spider [81], in 2017 and 2018 respectively. These are the first large-scale, multi-domain benchmarks that made it possible to train and evaluate neural text-to-SQL systems and provided a common tool to compare different systems easily. While other benchmarks have followed (e.g., [9, 24, 44, 49, 84]), these two remain the most popular ones. ┘

The first neural NLDBs (e.g., [75, 86]) could generate simple queries over single tables of the WikiSQL dataset. Recent systems [7, 27, 68] can generate complex SQL queries over relational databases and achieve high performance scores on Spider. Schema linking is not always an explicit component of a neural NLDB. Systems such as Seq2SQL [86], SQLNet [75], and HydraNet [52] do not perform schema linking, relying mainly on the neural network to correctly translate the NL question over the underlying data. However, it is not clear whether pre-trained neural architectures defy the need for schema linking. On the other hand, there is little evidence on how fast and scalable schema linking approaches are, especially for very large databases, with the exception of DBTagger [66] that provides experimental insights into the time and memory requirements of its schema linking approach.

Most systems do not ensure that the generated SQL is syntactically correct (e.g., Seq2SQL [86], SQLNet [75], HydraNet [52], IncSQL [61], RyanSQL [10]). Output refinement can be applied on a trained model to avoid producing incorrect SQL queries. However, it adds an additional burden to the system and increases the time needed to generate a SQL query. Its effectiveness versus the incurred overhead are yet to be studied.

The remarkable rise of neural NLDBs has not been followed by their widespread adoption in commercial products yet. There are many obstacles on the way to deliver the promise of truly enabling accessing data using natural language. A significant one is their view of the text-to-SQL problem as a language translation problem and their focus on translation accuracy over a specific dataset [29]. This is one side of the coin though, as we explain next.

## **6 Not (just) a Language Translation Problem: The Role of SQL and the Database Schema**

Even if a system eliminated all linguistic problems and could perfectly understand a NL query, additional challenges stem from the *SQL expressivity* and the *schema of the data*.

The language SQL was developed by IBM, as a human-friendly way to query relational data. However, SQL is a structured language with a strict grammar, which leads to limited expressivity when compared to natural language or other programming languages. For

example, it lacks constructs for iteration and recursion, amongst other things. There are queries that are easy to express in natural language, but their respective SQL is more complex and less intuitive. For example, the query “Return the movie with the best rating” may be mapped to a nested SQL query. While the original NL query is simple, building the complex SQL query may be challenging for the system. Full natural language is more expressive than single SQL queries. There are computationally reasonable queries that might be expressed in NL, but which cannot be answered by a single SQL query. Finally, while a sentence in natural language may contain mistakes, and still be understood by a human, a SQL query needs to be syntactically and semantically correct to be executable over the underlying database.

The database schema also poses challenges. As one instance, a query like “Who is the director of *Beautiful Mind*” may hide implicit join operations due to database normalization. Moreover, similar NL queries may need different processing on two different database schemas. For example, in a university database, every person is either a Student or a Faculty member, so these comprise two relations. On the other hand, movies have several genres that cannot be stored as different tables. They are stored in a Genre relation and are connected with movies through a many-to-many relationship. As a result, similar queries, such as “comedies released in 2018” and “students enrolled in 2018” are handled differently. In the former case, the system maps “comedies” to a value in the Genre table and joins it with the Movie table whereas it just maps “students” to the Student relation in the latter case.

All these challenges make the text-to-SQL problem challenging. Not only it is difficult to understand a NL query but it is also difficult to build the correct SQL query. Perhaps even more critically, similar questions may lead to a different outcome over different databases: one question may be successfully translated over one database and the other may not, due to issues such as ambiguity, paraphrasing, and different database schemas. Neural approaches addressing the problem as purely a language translation one ignore the particularities of the problem that stem from SQL and the database schema. Not surprisingly though, database approaches that view the text-to-SQL problem as mainly a database mapping one, also fall short in appreciating its multi-dimensional nature. Both perspectives, neural and database, have their merits to consider along their shortcomings in the quest of better NLIDBs.

## 7 Where we Stand: Limitations and Lessons Learnt

The systems that approach the text-to-SQL as a database mapping problem provide a more grounded solution that leverages the underlying data and relationships to interpret the NL question and build an executable SQL query. They also include explicit methods for dealing with query ambiguity both at the level of query terms as well as at the level of the query. However, existing approaches struggle with more complex and diverse NL queries and cannot easily cope with NL challenges, such as synonyms, paraphrasing and typos [25].

On the other hand, neural NLIDBs, often completely ignoring the underlying data, promise to be more generalizable both in terms of the different types of NL queries the methods can understand as well as the different databases they can work on thanks to their extensive training. However, in practice, existing approaches focus on limited-scope problems and their accuracy severely degrades with more complex and diverse NL and SQL queries as well as complex databases [39]. They depend on training data and cannot cope with unseen databases and queries. For example, Spider [81], which is very popular for training and evaluating text-to-SQL systems, contains queries over 200 relational databases from 138 different domains. These are toy databases with simple schemas and small sizes not resembling real-world databases. Moreover, most neural approaches support size-limiting

input representations that cannot possibly leverage the wealth of a real-world database comprising hundreds of tables and attributes. These limitations become highly relevant when applying a text-to-SQL system to an actual database [29] used in a business, research or any other real-world use case. Such databases can pose difficulties not encountered in the datasets used to train and evaluate such systems, for example, a large number of tables and attributes and table and column names that use domain-specific terminology. Last but not least, models used so far are typically quite large, questioning their practical use <sup>1</sup>.

## 8 The Challenges of NLIDBs

The challenges coming from the NL side and the challenges from the SQL and database side combined create a set of novel and unique to NLIDBs challenges that haunt current efforts.

### Query answerability: How to tell if the NL question can be answered or not and why.

For SQL queries, the answerability question can be answered deterministically based on two checks: syntactic and semantic correctness. When a database receives a SQL query, the parser checks whether the query adheres to the SQL syntax (*syntactic parsing*) and whether it contains tables and columns that exist in the underlying database (*semantic parsing*). Both checks are easy, fast, and conclusive. If the query contains any (syntactic or semantic) error, the system stops query processing and informs the user of the exact problem that makes the query unanswerable. The user can correct the query accordingly. In a NLIDB, the query answerability question is more convoluted: *does the NL question map to SQL that captures the query intent and is executable over the underlying database?* It is hard to define a set of comprehensive checks needed but it is also hard to answer them always successfully.

Two important checks are still (a) semantic correctness, meaning that the concepts mentioned in the query can be mapped to database columns and tables, and (b) syntactic correctness of the generated SQL. In database approaches to NLIDBs, the entity mapper performs the semantic parsing while the interpretation generator is responsible for the syntactic parsing since it makes sure that syntactically correct SQL queries are generated. Due to issues such as the language ambiguity, and the linguistic and conceptual gap between how the user formulates the query and how the system stores the data, *NLIDBs may fail to recognize that a question is answerable*. For instance, for the query “*How many people live in Amsterdam?*”, a NLIDB may not link the *population* attribute in the database schema to “*How many people live*”. A NLIDB may also fail to recognize that a NL question cannot be answered, and still try to generate a SQL query. These problems are even more pronounced in neural NLIDBs. Especially those that do not employ schema linking and output refinement techniques are more prone to generate non-executable queries.

For NL questions, there have been some attempts to define query answerability devising categories of unanswerable questions [69, 85]. For example, the *calculation unanswerable category* requires mapping the concept mentioned in the user question to composite operations over existing table columns that are not known SQL operations or even user-defined functions [69]. The *out-of-scope category* means that the question is out of SQL’s operation scope, such as when the user requests for charts [69]. *Improper to DB* refers to questions such as small talk or asking-opinion questions that are not proper to any databases [85].

Neural end-to-end parsing models ignore modeling questions in a fine-grained manner, which results in an inability to precisely detect and locate the specific reasons for unanswerable questions [69]. Furthermore, most existing text-to-SQL datasets used for training lack

---

<sup>1</sup> The training cost as well as the energy consumption [60] of such big models are important concerns.

ambiguous and unanswerable questions (e.g. Spider [81]) or if they contain, they are not marked in a way that the system can recognize them and learn accordingly (e.g., WikiSQL [86]). A NLIDB parsing system should detect answerable and unanswerable questions, and it should locate the specific reasons and generate corresponding explanations to guide the user in rectification. To that end, some recent efforts have emerged [82, 69]. For example, a weakly supervised DTE (Detecting-Then-Explaining) model for error detection, localization, and explanation [69] is trained on a dataset called TRIAGESQL [85].

**Query coverage: What is the set of NL questions allowed.** A NLIDB’s query capabilities are not obvious to the user [25, 64]. Users find it difficult to understand what kinds of questions the system can or cannot cope with. It is often not clear to the user whether a rejected question is outside the system’s query coverage (what types of NL statements it can recognize as queries that can be mapped to SQL), or whether it is outside the system’s conceptual coverage (how the data is actually stored). Users are often forced to rephrase their questions, until a form the system can understand is reached. In other cases, users never try questions the NLIDB could in principle handle, because they think the questions are outside the subset of natural language supported by the NLIDB. To frame the query coverage of the system, some NLIDBs explicitly restrict the set of natural language expressions the user is allowed to input, so that the limits of this subset are obvious to the user (e.g., the early PRE [21], SODA [4] and ExpressQ [83]). Clearly, there is a trade-off for an NLIDB between: (i) the usability and expressivity of natural language, and (ii) the reduction of ambiguity by imposing a more structured input syntax, which may lead to higher effectiveness.

In order to understand and expand the query coverage of NLIDBs, benchmarks play a critical role. Existing ones fail to address the question of what type of NL and SQL queries a system can understand and build, respectively. One important reason for that is the lack of a query categorization. Spider [81] has four very coarse-grained classes of queries. The first organized effort to understanding the query coverage of a NLIDB is a query benchmark [25] consisting of keyword and natural language queries over three datasets, divided into 17 categories that aim to test the systems on specific linguistic and SQL aspects of the problem of translating free-form queries to SQL. The authors also provide a full experimental methodology that studies both the effectiveness (correctness of answers), and the efficiency (execution time) of the systems.

Other benchmarks focus on testing specific query capabilities. For instance, Spider-DK [24] extends Spider to explore system capabilities at cross-domain generalization (i.e., robustness to domain-specific vocabulary across different domains), while Spider-Syn [23] focuses on robustness to synonyms and different vocabulary. TRIAGESQL [85] focuses on the answerability problem and defines four types of unanswerable questions along with answerable questions. Clearly, more effort is needed in coming up with benchmarks that can provide clear insights into query coverage.

There are several types of queries that current benchmarks do not cover and neural systems are not trained to answer. For instance, *meta-knowledge questions* are questions referring to knowledge about knowledge (e.g., ASK [65]), such as “*What information is in the database?*”, “*What is known about ships?*” [62] or “*What are the possible employee job titles?*” In *modal questions*, the user asks whether something can or must be the case. For example: “*Can a female employee work in sales?*” Furthermore, NLIDBs cannot answer temporal questions [3] because they cannot cope with the semantics of natural language temporal expressions (e.g. tenses/aspects, temporal subordinators), and they were designed to interface to “snapshot” databases, that do not facilitate the manipulation of time-dependent information.

**Domain portability: How to cope with new domain knowledge.** Early NLIDBs were each designed for a particular database application. Lunar [74], for example, was designed to support English questions, referring to a database of a particular structure, holding data about moon rocks. Such application-tailored NLIDBs were very difficult to port to different applications or databases. Different to the built-in formal query language interpreters that commercial database systems come with, more recent NLIDBs usually require tedious configuration phases before they can be used for a different database or domain. A knowledge engineer needs to capture the domain knowledge, such as rules, knowledge bases or ontologies (e.g., CLE [2], SODA [4], ATHENA [57]). The db administrator needs to create inverted indexes (e.g., TEAM [26], DISCOVER [33]). Neural systems cannot make good predictions for unseen NL questions and domains unless trained.

The top neural NLIDBs achieve high accuracy numbers on Spider. However, the majority of databases in the Spider benchmark are of low complexity, and contain general knowledge. Furthermore, most queries are simple without complex structures or operations. Neural machine translation systems pre-trained on common knowledge datasets, like Spider, typically fail in complex domains due to the large disparity in subject matter [84]. Real-world databases often store large amounts of data with hundreds of attributes. These attributes may have non-descriptive names or store numerical measurements (an example is the astrophysics database called Sloan Digital Sky Survey (SDSS)<sup>2</sup>). Learning the mapping of a token from a natural language question to the relevant database attribute may necessitate additional training as well as ontologies that describe the meaning of attributes and tables. On top of that, domain-specific queries may be more elaborate containing functions and mathematical operators between attributes.

Given the difficulties that arise when creating an NL interface for a real-world database, one approach is to build specialized, domain-specific benchmarks for training and evaluating such systems. For instance, Spider-DK [24] extends Spider to explore system capabilities at cross-domain generalization (i.e., robustness to domain-specific vocabulary across different domains). BIRD [49] is a dataset with questions over large-scale databases that can better represent real use-case scenarios. ScienceBenchmark [84] focuses on three real-world, highly domain-specific databases. EHRSQL [45] contains questions over two databases related to health records. Manually crafting a benchmark for a new domain requires domain-specific expertise and is challenging due to the volume of data needed. Hence, an alternative approach is *data augmentation*, i.e. automatic benchmark generation [84].

**Verification: How to tell if the system response matches the NL question.** A NLIDB may generate queries in its output that contain errors and do not match the NL question's intent. For example, the output SQL may contain wrong columns or tables, values that are not found in the data or they are found in a different form, unnecessary or wrong joins, and so forth. Even when an NLIDB cannot understand a question or the query is not answerable, the system may still try to generate a SQL query (as most neural systems do). Another problem is that NL questions often have several readings, and the system may select a reading of a question that is different from the reading the user had in mind. In these cases, it may be hard for the user to understand that the system has actually answered a different question. To avoid such misunderstandings, TQA [13] was a very early system that contained a module that converted the SQL query back to natural language. Query explanations in natural language provide a means for users to cross-check their question to the explanations of the predicted SQL queries and validate the results [20, 40, 71].

---

<sup>2</sup> <https://www.sdss.org/>



**Efficiency.** For a NLIDB, focusing on improving its query answering capabilities is only one side of the coin. Its response time also matters for their adoption. It consists of two parts: the *NL translation time* and the *SQL execution time*.

The NL translation time is an overhead to the overall query execution time that the user will experience, and hence needs to be optimized. Unfortunately, current systems overlook the significance of optimized execution, and employ methods that are time-consuming and do not scale well to large databases. Neural NLIDBs typically rely on very complex models. While the use of large PLMs usually translates to higher accuracy, it also translates to higher inference times. Output refinement techniques are also adding extra overhead. For example, one of the best-performing models on the Spider dataset, T5-3B+PICARD, uses a large PLM along with a computationally-intensive output refinement technique [58]. Schema linking also contributes to the overall translation time. These techniques have been shown to be beneficial for systems working on the Spider dataset, but they have yet to be tested on large-scale databases, where their overhead may be significant. Even though using indices and other DB lookup techniques might speed up schema linking, it is still questionable if looking up multiple words or n-grams for every NLQ, is efficient in a real application.

The SQL execution time is also important. A NL question may be written in SQL in many equivalent but not equally efficient ways. It is the NLIDB's responsibility to choose the most efficient one. Current neural systems only focus on the translation. Early text-to-SQL systems originating from the database community [32, 33, 47, 51, 83] not only tried to generate correct SQL queries but also optimal in terms of execution speed. Hence, many of them contained logic for generating code that would return the desired results fast.

A NLIDB should have a fast response time, even when the question cannot be interpreted. Ultimately, allowing the user to express questions in natural language should free them from the technical details of how this query should be expressed in the underlying system language and how it should be executed efficiently.

**Reasoning.** Most NLIDBs are direct interfaces to the underlying database, in the sense that they simply translate user questions to suitable database queries. In some cases, it may not be possible to answer a natural language question, although all the necessary raw data are present in the database. Questions involving common sense or domain expertise are typical examples. In these cases, to produce the answer, the NLIDB must be able to carry out reasoning based on the data stored in the database [3].

## 9 Looking Forward: From SQL to NQL

In practice, no modern DBMS comes with an integrated NL query language, nor does exist a NL client that connects to a database seamlessly like an SQL client and allows a user to pose queries in NL. Nevertheless, with the galloping progress of deep learning methods, the emergence of LLMs and vector databases, and several other developments, many researchers go as far as to envision that NQL (Natural Query Language) will replace SQL. In any scenario, below, we discuss some requirements for a NQL (Natural Query Language) that open up several fascinating research directions. Interestingly, such requirements have been discussed in early systems before the advent of neural NLIDBs (e.g., [63]).

**R1. Query expressivity:** Using a query language such as SQL, the user knows exactly what queries are possible. In a similar vein, the set of NL queries that a NLIDB supports should be clearly defined so that a user is aware of the available query capabilities.

**R2. Data independence:** A NLIDB should support the same query expressivity for different databases. In other words, the same type of NL query should be possible over any database. For example, if the user could ask “what is the average X of Y” in one database, then this type of query should be possible in any other database.

**R3. Performance:** The system should transparently find the most efficient way to answer a NL query, minimizing both the translation overhead and the query execution cost.

**R4. Scalability:** A NLIDB should be feasible and scalable over any database.

Requirement R1 is important because up to now almost none of the known text-to-SQL systems provides a clearly defined query language or specification of its query capabilities. For the user, it is a trial-and-error process to see what queries can be understood and answered by the system. Is it possible to come up with a query language specification that systems can refer to in order to describe their query expressivity?

Towards R1, a query categorization in the spirit of [25] may be a good starting point. This could enable the creation of appropriate benchmarks for the comparison of the query capabilities of different systems. Even devising an appropriate query categorization and an appropriate benchmark raises several challenges: what categories to choose, what queries should be in each category, which datasets to use. Furthermore, one should take into account SQL equivalence (different SQL queries that return the same results), and NL ambiguity (a NL query may have more than one correct translation over the data).

Requirement R2 complements R1 in saying that the same query expressivity should be supported over any database. This comes naturally with query languages such as SQL. For instance, SPJ queries can be supported over any data. For a NLIDB, that does not hold. Going from one database to another, the same type of queries may not be supported. As we have already pointed out, this is a major concern for neural systems.

One could build specialized, domain-specific benchmarks for training and evaluating text-to-SQL systems for a new database. Manually crafting such benchmarks is time-consuming. Data augmentation, i.e., automatic benchmark generation, is an open research direction [73]. However, benchmarks provide a means to demonstrate query expressivity. How does one ensure data independence is a different beast and finding better training datasets is not the solution to the problem. Rethinking the system design is needed instead.

Towards this direction, approaches that have been proposed by the database community have been shown to be more effective from the data independence perspective, since they rely on the information that the database provides. This potentially points to the need of re-thinking our approach to the text-to-SQL problem. Some parts of the solution may require DB methods to ensure data independence and some other parts may use neural models to generalize system knowledge, for example on the diversity and complexity of NL queries. How would a system that combines such capabilities look like?

Requirement R3 is about making NQL queries efficient. While the state-of-the-art systems are still dealing with “getting the answer right”, they are mostly overlooking the “getting the answer fast”. Improving translation speed by building efficient methods is necessary. But this may not be enough. Text-to-SQL systems originating from the DB community not only tried to generate correct SQL queries but also optimal in terms of execution speed. Improving the overall NQL answering time, i.e., both the translation and the execution, opens up several research opportunities, from building more efficient models to mapping translation and execution to operators and building NQL query plans that can be optimized. In fact, implementing natural language query capabilities closer to the DBMS would open up several opportunities to leverage both worlds, the database and NLP, from NLQ optimization to learning and improving the system’s query capabilities.



R4 highlights the need for realistic solutions. Deep learning text-to-SQL systems typically rely on very complex models, which have been trained and evaluated on toy databases (contained in existing benchmarks). In several cases, it may not be possible to have the required resources to train such enormous models. Furthermore, since these models require that the database schema is given as input, they do not scale well to very large databases, with hundreds of attributes and tables (such as astrophysics and biological data). Instead of focusing on increasing the model complexity aiming at translation accuracy, we need to design solutions that also take into account system efficiency, complexity, and scale.

## 10 Conclusion

Querying data in natural language has been the holy grail of the database community for several decades. Several efforts ended up in frustrated users with unmet expectations and disappointed researchers and developers. Commercial products were given up. However, the landscape has changed. On the one hand, technologies evolve and become increasingly more powerful. On the other hand, people are becoming accustomed to interacting with devices and software using natural language. In a few years, the way we interact with data will probably be very different from what we know nowadays.

---

### References

- 1 Katrin Affolter, Kurt Stockinger, and Abraham Bernstein. A comparative survey of recent natural language interfaces for databases. *CoRR*, abs/1906.08990, 2019. doi:10.48550/arXiv.1906.08990.
- 2 Hiyan Alshawi, David M. Carter, Jan van Eijck, Robert C. Moore, Douglas B. Moran, and Stephen G. Pulman. Overview of the core language engine. In *Proceedings of the International Conference on Fifth Generation Computer Systems, FGCS 1988, Tokyo, Japan, November 28-December 2, 1988*, pages 1108–1115. OHMSHA Ltd. Tokyo and Springer-Verlag, 1988.
- 3 Ion Androutsopoulos, Graeme D. Ritchie, and Peter Thanisch. Natural language interfaces to databases – An introduction. *Natural Language Engineering*, 1(1):29–81, 1995. doi:10.1017/S135132490000005X.
- 4 Lukas Blunschi, Claudio Jossen, Donald Kossmann, Magdalini Mori, and Kurt Stockinger. SODA: Generating sql for business users. *PVLDB*, 5(10):932–943, 2012. doi:10.14778/2336664.2336667.
- 5 Ben Bogin, Jonathan Berant, and Matt Gardner. Representing schema structure with graph neural networks for text-to-SQL parsing. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4560–4565, Florence, Italy, jul 2019. Association for Computational Linguistics. doi:10.18653/v1/P19-1448.
- 6 Ben Bogin, Matt Gardner, and Jonathan Berant. Global reasoning over database structures for text-to-SQL parsing. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3657–3662, Hong Kong, China, nov 2019. Association for Computational Linguistics. doi:10.18653/v1/D19-1378.
- 7 Ursin Brunner and Kurt Stockinger. Valuenet: A neural text-to-sql architecture incorporating values. *arXiv*, abs/2006.00888, 2020. doi:10.48550/arXiv.2006.00888.
- 8 Ruichu Cai, Boyan Xu, Zhenjie Zhang, Xiaoyan Yang, Zijian Li, and Zhihao Liang. An encoder-decoder framework translating natural language to database queries. In Jérôme Lang, editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI*, pages 3977–3983. ijcai.org, 2018. doi:10.24963/ijcai.2018/553.

- 9 Shuaichen Chang, Jun Wang, Mingwen Dong, Lin Pan, Henghui Zhu, Alexander Hanbo Li, Wuwei Lan, Sheng Zhang, Jiarong Jiang, Joseph Lilien, et al. Dr. spider: A diagnostic evaluation benchmark towards text-to-sql robustness. *arXiv preprint*, 2023. doi:10.48550/arXiv.2301.08881.
- 10 DongHyun Choi, Myeong Cheol Shin, EungGyun Kim, and Dong Ryeol Shin. Ryansql: Recursively applying sketch-based slot fillings for complex text-to-sql in cross-domain databases, 2020. doi:10.48550/arXiv.2004.03125.
- 11 E. F. Codd. Seven steps to rendezvous with the casual user. In J. W. Klimbie and K. L. Koffeman, editors, *Data Base Management, Proceeding of the IFIP Working Conference Data Base Management, Cargèse, Corsica, France, April 1-5, 1974*, pages 179–200. North-Holland, jan 1974.
- 12 Ann A. Copestake and Karen Sparck Jones. Natural language interfaces to databases. *Knowl. Eng. Rev.*, 5(4):225–249, 1990. doi:10.1017/S0269888900005476.
- 13 Fred Damerou. Problems and some solutions in customization of natural language database front ends. *ACM Trans. Inf. Syst.*, 3(2):165–184, 1985. doi:10.1145/3914.3915.
- 14 Marie-Catherine de Marneffe, Bill MacCartney, and Christopher D. Manning. Generating typed dependency parses from phrase structure parses. *LREC*, pages 449–454, 2006. URL: <http://www.lrec-conf.org/proceedings/lrec2006/summaries/440.html>.
- 15 A.N. De Roeck. *A Natural Language System Based on Formal Semantics*. Universiti Sains Malaysia, 1991. URL: <https://books.google.gr/books?id=hrtdnQAACAAJ>.
- 16 Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019. doi:10.48550/arXiv.1810.04805.
- 17 Li Dong and Mirella Lapata. Language to logical form with neural attention, 2016. doi:10.48550/arXiv.1601.01280.
- 18 Timothy Dozat and Christopher D. Manning. Deep biaffine attention for neural dependency parsing, 2017. doi:10.48550/arXiv.1611.01734.
- 19 Timothy Dozat and Christopher D. Manning. Simpler but more accurate semantic dependency parsing, 2018. doi:10.48550/arXiv.1807.01396.
- 20 Stavroula Eleftherakis, Orest Gkini, and Georgia Koutrika. Let the database talk back: Natural language explanations for SQL. In *Proceedings of the 2nd Workshop on Search, Exploration, and Analysis in Heterogeneous Datastores (SEA-Data 2021) co-located with 47th International Conference on Very Large Data Bases (VLDB 2021), Copenhagen, Denmark, August 20, 2021*, volume 2929 of *CEUR Workshop Proceedings*, pages 14–19. CEUR-WS.org, 2021. URL: <https://ceur-ws.org/Vol-2929/paper3.pdf>.
- 21 Samuel S. Epstein. Transportable natural language processing through simplicity - the PRE system. *ACM Trans. Inf. Syst.*, 3(2):107–120, 1985. doi:10.1145/3914.3985.
- 22 Han Fu, Chang Liu, Bin Wu, Feifei Li, Jian Tan, and Jianling Sun. Catsql: Towards real world natural language to SQL applications. *Proc. VLDB Endow.*, 16(6):1534–1547, 2023. doi:10.14778/3583140.3583165.
- 23 Yujian Gan, Xinyun Chen, Qiuping Huang, Matthew Purver, John R. Woodward, Jinxia Xie, and Pengsheng Huang. Towards robustness of text-to-SQL models against synonym substitution. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2505–2515, Online, aug 2021. Association for Computational Linguistics. doi:10.18653/v1/2021.acl-long.195.
- 24 Yujian Gan, Xinyun Chen, and Matthew Purver. Exploring underexplored limitations of cross-domain text-to-sql generalization. *arXiv preprint*, 2021. doi:10.48550/arXiv.2109.05157.
- 25 Orest Gkini, Theofilos Belmpas, Georgia Koutrika, and Yannis E. Ioannidis. An in-depth benchmarking of text-to-sql systems. In Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava, editors, *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, pages 632–644. ACM, 2021. doi:10.1145/3448016.3452836.

- 26 Barbara J. Grosz. TEAM: A transportable natural-language interface system. In *1st Applied Natural Language Processing Conference, ANLP 1983, Miramar-Sheraton Hotel, Santa Monica, California, USA, February 1-3, 1983*, pages 39–45. ACL, 1983. doi:10.3115/974194.974201.
- 27 Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. Towards complex text-to-sql in cross-domain database with intermediate representation, 2019. doi:10.48550/arXiv.1905.08205.
- 28 Larry R. Harris. Experience with INTELLECT: artificial intelligence technology transfer. *AI Mag.*, 5(2):43–50, 1984. URL: <https://ojs.aaai.org/index.php/aimagazine/article/view/437>.
- 29 Moshe Hazoom, Vibhor Malik, and Ben Bogin. Text-to-SQL in the wild: A naturally-occurring dataset based on stack exchange data. In *1st Workshop on Natural Language Processing for Programming (NLP4Prog 2021)*, pages 77–87, aug 2021.
- 30 Pengcheng He, Yi Mao, Kaushik Chakrabarti, and Weizhu Chen. X-sql: reinforce schema representation with context, 2019. doi:10.48550/arXiv.1908.08113.
- 31 Gary G. Hendrix, Earl D. Sacerdoti, Daniel Sagalowicz, and Jonathan Slocum. Developing a natural language interface to complex data. *ACM Trans. Database Syst.*, 3(2):105–147, jun 1978. doi:10.1145/320251.320253.
- 32 Vagelis Hristidis, Luis Gravano, and Yannis Papakonstantinou. Efficient IR-style keyword search over relational databases. In *VLDB*, pages 850–861, 2003. doi:10.1016/B978-012722442-8/50080-X.
- 33 Vagelis Hristidis and Yannis Papakonstantinou. Discover: Keyword search in relational databases. In *VLDB*, pages 670–681, 2002. doi:10.1016/B978-155860869-6/50065-2.
- 34 Binyuan Hui, Xiang Shi, Ruiying Geng, Binhua Li, Yongbin Li, Jian Sun, and Xiaodan Zhu. Improving text-to-sql with schema dependency learning, 2021. doi:10.48550/arXiv.2103.04399.
- 35 Wonseok Hwang, Jinyeong Yim, Seunghyun Park, and Minjoon Seo. A comprehensive exploration on wikisql with table-aware word contextualization, 2019. doi:10.48550/arXiv.1902.01069.
- 36 Tim Johnson. Natural language computing: The commercial applications. *Knowl. Eng. Rev.*, 1(3):11–23, 1984. doi:10.1017/S0269888900000588.
- 37 George Katsogiannis-Meimarakis and Georgia Koutrika. A deep dive into deep learning approaches for text-to-sql systems. In *Proceedings of the 2021 International Conference on Management of Data, SIGMOD/PODS '21*, pages 2846–2851, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3448016.3457543.
- 38 George Katsogiannis-Meimarakis and Georgia Koutrika. A survey on deep learning approaches for text-to-sql. *The VLDB Journal*, 2023. doi:10.1007/s00778-022-00776-8.
- 39 Hyeonji Kim, Byeong-Hoon So, Wook-Shin Han, and Hongrae Lee. Natural language to sql: Where are we today? *Proc. VLDB Endow.*, 13(10):1737–1750, 2020. doi:10.14778/3401960.3401970.
- 40 Andreas Kokkalis, Panagiotis Vagenas, Alexandros Zervakis, Alkis Simitsis, Georgia Koutrika, and Yannis E. Ioannidis. Logos: a system for translating queries into narratives. In K. Selçuk Candan, Yi Chen, Richard T. Snodgrass, Luis Gravano, and Ariel Fuxman, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 673–676. ACM, 2012. doi:10.1145/2213836.2213929.
- 41 Georgia Koutrika, Alkis Simitsis, and Yannis E. Ioannidis. Précis: The essence of a query answer. In Ling Liu, Andreas Reuter, Kyu-Young Whang, and Jianjun Zhang, editors, *Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006, 3-8 April 2006, Atlanta, GA, USA*, pages 69–78. IEEE Computer Society, 2006. doi:10.1109/ICDE.2006.114.
- 42 Jayant Krishnamurthy, Pradeep Dasigi, and Matt Gardner. Neural semantic parsing with type constraints for semi-structured tables. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1516–1526, Copenhagen, Denmark, sep 2017. Association for Computational Linguistics. doi:10.18653/v1/D17-1160.

- 43 John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning, ICML '01*, pages 282–289, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.
- 44 Chia-Hsuan Lee, Oleksandr Polozov, and Matthew Richardson. Kaggledbqa: Realistic evaluation of text-to-sql parsers. *arXiv preprint*, 2021. doi:10.48550/arXiv.2106.11455.
- 45 Gyubok Lee, Hyeonji Hwang, Seongsu Bae, Yeonsu Kwon, Woncheol Shin, Seongjun Yang, Minjoon Seo, Jong-Yeup Kim, and Edward Choi. Ehrsql: A practical text-to-sql benchmark for electronic health records. *Advances in Neural Information Processing Systems*, 35:15589–15601, 2022. URL: [http://papers.nips.cc/paper\\_files/paper/2022/hash/643e347250cf9289e5a2a6c1ed5ee42e-Abstract-Datasets\\_and\\_Benchmarks.html](http://papers.nips.cc/paper_files/paper/2022/hash/643e347250cf9289e5a2a6c1ed5ee42e-Abstract-Datasets_and_Benchmarks.html).
- 46 Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension, 2019. doi:10.48550/arXiv.1910.13461.
- 47 Fei Li and H. V. Jagadish. Constructing an interactive natural language interface for relational databases. *PVLDB*, 8(1):73–84, sep 2014. doi:10.14778/2735461.2735468.
- 48 Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. RESDSQL: decoupling schema linking and skeleton parsing for text-to-sql. In Brian Williams, Yiling Chen, and Jennifer Neville, editors, *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023*, pages 13067–13075. AAAI Press, 2023. doi:10.1609/aaai.v37i11.26535.
- 49 Jinyang Li, Binyuan Hui, Ge Qu, Binhua Li, Jiayi Yang, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, et al. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *arXiv preprint*, 2023. doi:10.48550/arXiv.2305.03111.
- 50 Xi Victoria Lin, Richard Socher, and Caiming Xiong. Bridging textual and tabular data for cross-domain text-to-SQL semantic parsing. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4870–4888, Online, nov 2020. Association for Computational Linguistics. doi:10.18653/v1/2020.findings-emnlp.438.
- 51 Yi Luo, Xuemin Lin, Wei Wang, and Xiaofang Zhou. Spark: Top-k keyword query in relational databases. In *ACM SIGMOD*, pages 115–126, 2007. doi:10.1145/1247480.1247495.
- 52 Qin Lyu, Kaushik Chakrabarti, Shobhit Hathi, Souvik Kundu, Jianwen Zhang, and Zheng Chen. Hybrid ranking network for text-to-sql. Technical Report MSR-TR-2020-7, Microsoft Dynamics 365 AI, mar 2020. URL: <https://www.microsoft.com/en-us/research/publication/hybrid-ranking-network-for-text-to-sql/>.
- 53 Nikolaus Ott. Aspects of the automatic generation of SQL statements in the natural language query interface. *Inf. Syst.*, 17(2):147–159, 1992. doi:10.1016/0306-4379(92)90009-C.
- 54 Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL: <http://www.aclweb.org/anthology/D14-1162>, doi:10.3115/v1/d14-1162.
- 55 Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 2020. doi:10.48550/arXiv.1910.10683.
- 56 Ohad Rubin and Jonathan Berant. SmBoP: Semi-autoregressive bottom-up semantic parsing. In *Proceedings of the 5th Workshop on Structured Prediction for NLP (SPNLP 2021)*, pages 12–21, Online, aug 2021. Association for Computational Linguistics. doi:10.18653/v1/2021.spnlp-1.2.

- 57 Diptikalyan Saha, Avriela Floratou, Karthik Sankaranarayanan, Umar Farooq Minhas, Ashish R. Mittal, Fatma Özcan, IBM Research. Bangalore, and IBM Research. Almaden. *ATHENA: An Ontology-Driven System for Natural Language Querying over Relational Data Stores*. VLDB, 2016. URL: <http://www.vldb.org/pvldb/vol19/p1209-saha.pdf>, doi:10.14778/2994509.2994536.
- 58 Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. Picard: Parsing incrementally for constrained auto-regressive decoding from language models, 2021. doi:10.48550/arXiv.2109.05093.
- 59 Jaydeep Sen, Chuan Lei, Abdul Quamar, Fatma Özcan, Vasilis Efthymiou, Ayushi Dalmia, Greg Stager, Ashish R. Mittal, Diptikalyan Saha, and Karthik Sankaranarayanan. ATHENA++: natural language querying for complex nested SQL queries. *Proc. VLDB Endow.*, 13(11):2747–2759, 2020. URL: <http://www.vldb.org/pvldb/vol13/p2747-sen.pdf>.
- 60 Or Sharir, Barak Peleg, and Yoav Shoham. The cost of training NLP models: A concise overview. *CoRR*, abs/2004.08900, 2020. doi:10.48550/arXiv.2004.08900.
- 61 Tianze Shi, Kedar Tatwawadi, Kaushik Chakrabarti, Yi Mao, Oleksandr Polozov, and Weizhu Chen. Incsql: Training incremental text-to-sql parsers with non-deterministic oracles, 2018. doi:10.48550/arXiv.1809.05054.
- 62 Alkis Simitsis, Georgia Koutrika, and Yannis Ioannidis. Précis: from unstructured keywords as queries to structured databases as answers. *The VLDB Journal*, 17(1):117–149, 2008. doi:10.1007/s00778-007-0075-9.
- 63 Marjorie Templeton and John Burger. Problems in natural-language interface to DBMS with examples from EUFID. In *First Conference on Applied Natural Language Processing*, pages 3–16, Santa Monica, California, USA, feb 1983. Association for Computational Linguistics. doi:10.3115/974194.974197.
- 64 Harry R. Tennant, Kenneth M. Ross, Richard M. Saenz, Craig W. Thompson, and James R. Miller. Menu-based natural language understanding. In Mitchell P. Marcus, editor, *21st Annual Meeting of the Association for Computational Linguistics, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, June 15-17, 1983*, pages 151–158. ACL, 1983. doi:10.3115/981311.981341.
- 65 Bozena Henisz Thompson and Frederick B. Thompson. Introducing ask, A simple knowledgeable system. In *1st Applied Natural Language Processing Conference, ANLP 1983, Miramar-Sheraton Hotel, Santa Monica, California, USA, February 1-3, 1983*, pages 17–24. ACL, 1983. doi:10.3115/974194.974198.
- 66 Arif Usta, Akifhan Karakayali, and Özgür Ulusoy. Dbtagger: Multi-task learning for keyword mapping in NLIDBs using bi-directional recurrent neural networks. *Proc. VLDB Endow.*, 14(5):813–821, jan 2021. doi:10.14778/3446095.3446103.
- 67 Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017. doi:10.48550/arXiv.1706.03762.
- 68 Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers, 2020. doi:10.48550/arXiv.1911.04942.
- 69 Bing Wang, Yan Gao, Zhoujun Li, and Jian-Guang Lou. Know what I don’t know: Handling ambiguous and unknown questions for text-to-sql. In Anna Rogers, Jordan L. Boyd-Graber, and Naoaki Okazaki, editors, *Findings of the Association for Computational Linguistics: ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 5701–5714. Association for Computational Linguistics, 2023. doi:10.18653/V1/2023.FINDINGS-ACL.352.
- 70 Chenglong Wang, Kedar Tatwawadi, Marc Brockschmidt, Po-Sen Huang, Yi Mao, Oleksandr Polozov, and Rishabh Singh. Robust text-to-sql generation with execution-guided decoding, 2018. arXiv:1807.03100.



- 71 Weiguo Wang, Sourav S. Bhowmick, Hui Li, Shafiq R. Joty, Siyuan Liu, and Peng Chen. Towards enhancing database education: Natural language generation meets query execution plans. In Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava, editors, *SIGMOD '21: International Conference on Management of Data*, pages 1933–1945. ACM, 2021. doi:10.1145/3448016.3452822.
- 72 David H. D. Warren and Fernando C. N. Pereira. An efficient easily adaptable system for interpreting natural language queries. *Am. J. Comput. Linguistics*, 8(3-4):110–122, 1982.
- 73 Nathaniel Weir, Prasetya Utama, Alex Galakatos, Andrew Crotty, Amir Ilkhechi, Shekar Ramaswamy, Rohin Bhushan, Nadja Geisler, Benjamin Hättasch, Steffen Eger, Ugur Çetintemel, and Carsten Binnig. Dbpal: A fully pluggable NL2SQL training pipeline. In David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo, editors, *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, pages 2347–2361. ACM, 2020. doi:10.1145/3318464.3380589.
- 74 William A. Woods. Procedural semantics for a question-answering machine. In *Proceedings of the AFIPS '68 Fall Joint Computer Conference, December 9-11, 1968, San Francisco, California, USA - Part I*, volume 33, pages 457–471, 1968. doi:10.1145/1476589.1476653.
- 75 Xiaojun Xu, Chang Liu, and Dawn Song. Sqlnet: Generating structured queries from natural language without reinforcement learning, 2017. doi:10.48550/arXiv.1711.04436.
- 76 Kuan Xuan, Yongbo Wang, Yongliang Wang, Zujie Wen, and Yang Dong. Sead: End-to-end text-to-sql generation with schema-aware denoising, 2021. doi:10.48550/arXiv.2105.07911.
- 77 Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, and Thomas Dillig. Sqlizer: Query synthesis from natural language. *PACMPL*, pages 63:1–63:26, 2017. doi:10.1145/3133887.
- 78 Pengcheng Yin and Graham Neubig. A syntactic neural model for general-purpose code generation, 2017. doi:10.48550/arXiv.1704.01696.
- 79 Pengcheng Yin and Graham Neubig. TRANX: A transition-based neural abstract syntax parser for semantic parsing and code generation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 7–12, Brussels, Belgium, nov 2018. Association for Computational Linguistics. doi:10.18653/v1/D18-2002.
- 80 Tao Yu, Zifan Li, Zilin Zhang, Rui Zhang, and Dragomir Radev. Typesql: Knowledge-based type-aware neural text-to-sql generation, 2018. doi:10.48550/arXiv.1804.09769.
- 81 Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task, 2019. doi:10.48550/arXiv.1809.08887.
- 82 Wei Yu, Haiyan Yang, Mengzhu Wang, and Xiaodong Wang. Bravely say I don't know: Relational question-schema graph for text-to-sql answerability classification. *ACM Trans. Asian Low Resour. Lang. Inf. Process.*, 22(4):111:1–111:18, 2023. doi:10.1145/3579030.
- 83 Zhong Zeng, Mong Li Lee, and Tok Wang Ling. Answering keyword queries involving aggregates and groupby on relational databases. *EDBT*, pages 161–172, 2016. doi:10.5441/002/edbt.2016.17.
- 84 Yi Zhang, Jan Deriu, George Katsogiannis-Meimarakis, Catherine Kosten, Georgia Koutrika, and Kurt Stockinger. Sciencebenchmark: A complex real-world benchmark for evaluating natural language to sql systems, 2023. doi:10.48550/arXiv.2306.04743.
- 85 Yusen Zhang, Xiangyu Dong, Shuaichen Chang, Tao Yu, Peng Shi, and Rui Zhang. Did you ask a good question? A cross-domain question intention classification benchmark for text-to-sql. *CoRR*, abs/2010.12634, 2020. doi:10.48550/arXiv.2010.12634.
- 86 Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning, 2017. doi:10.48550/arXiv.1709.00103.

# How Database Theory Helps Teach Relational Queries in Database Education

Sudeepa Roy ✉ 

Duke University, Durham, NC, USA  
RelationalAI, Berkeley, CA, USA  
(Visiting Scientist)

Amir Gilad ✉ 

Hebrew University of Jerusalem, Israel

Yihao Hu ✉ 


Duke University, Durham, NC, USA

Hanze Meng ✉ 

Duke University, Durham, NC, USA

Zhengjie Miao ✉ 

Simon Fraser University, Burnaby, Canada

Kristin Stephens-Martinez ✉ 

Duke University, Durham NC, USA

Jun Yang ✉ 

Duke University, Durham, NC, USA

---

## Abstract

Data analytics skills have become an indispensable part of any education that seeks to prepare its students for the modern workforce. Essential in this skill set is the ability to work with structured relational data. Relational queries are based on logic and may be declarative in nature, posing new challenges to novices and students. Manual teaching resources being limited and enrollment growing rapidly, automated tools that help students debug queries and explain errors are potential game-changers in database education. We present a suite of tools built on the foundations of database theory that has been used by over 1600 students in database classes at Duke University, showcasing a high-impact application of database theory in database education.

**2012 ACM Subject Classification** Theory of computation → Database theory; Information systems → Data management systems; Information systems → Structured Query Language

**Keywords and phrases** Query Debugging, SQL, Relational Algebra, Relational Calculus, Database Education, Boolean Provenance

**Digital Object Identifier** 10.4230/LIPIcs.ICDT.2024.2

**Category** Invited Talk

**Funding** This work was partially supported by NSF awards IIS-2008107 and IIS-1552538.

**Acknowledgements** The authors are grateful to graduate and undergraduate students Alexander Bendeck, Stephen Chen, Tiangang Chen, Jeremy Cohen, Kevin Day, James Leong, Qiulin Li, James Lim, Jeffrey Luo, Allen Pan, Aanya Sanghavi, Sharan Sokhi, Aparimeya Taneja, and Zachary Zheng who contributed to building the tracing tool for SQL queries. This work was done when Zhengjie Miao (part of his PhD dissertation research) and Amir Gilad were at Duke University.

## Extended Abstract

In a world where decisions are increasingly driven by data, data analytics skills have become an indispensable part of any education that seeks to prepare its students for the modern workforce, in particular, in the multi-billion dollar and rapidly-growing data analytics industry [7]. Essential in this skill set is the ability to work with *structured or relational data* in tabular form – such data can be queried directly to yield useful insights, or transformed into other representations for additional analysis, model training, or visualization. The standard “tools of trade” for manipulating structured data include the venerable and ubiquitous SQL language as well as popular data manipulation libraries, e.g., `dplyr` for R, `DataFrame`



© Sudeepa Roy, Amir Gilad, Yihao Hu, Hanze Meng, Zhengjie Miao, Kristin Stephens-Martinez, and Jun Yang;

licensed under Creative Commons License CC-BY 4.0

27th International Conference on Database Theory (ICDT 2024).

Editors: Graham Cormode and Michael Shekelyan; Article No. 2; pp. 2:1–2:9



Leibniz International Proceedings in Informatics

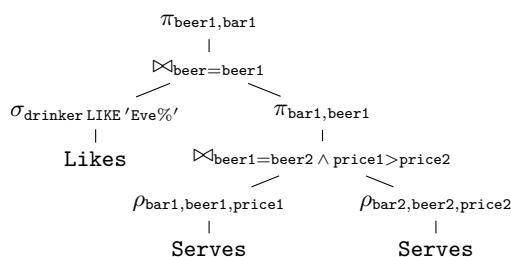
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

for Python *pandas*, and *Spark*. Despite differences in syntax, they are all fundamentally based on the *relational model* and rooted in relational query languages such as *Relational Calculus* (RC, similar to First Order Logic (FOL)) and *Relational Algebra* (RA). Learning and debugging relational queries, however, pose challenges to novices [2]. Even computer science students with programming background are often not used to thinking in terms of logic (e.g., when writing SQL queries) or functional programming (e.g., when writing queries using operators that resemble RA). In stark contrast to the plethora of educational tools for teaching traditional programming, there is a glaring lack of tools for helping novices learn and debug relational queries. The problem becomes more critical in a classroom setting, especially given the rapid growth of enrollment in database classes, and limited availability of manual help from instructors and teaching assistants in assisting students debug their queries. This motivates the need to build automated query debugging tools for database education that can give students succinct but comprehensive information about the mistakes they have made, and ideally also offer them advice or hints on how to resolve the mistakes without giving out the entire correct query.

Building tools for verifying, debugging, and fixing relational queries requires foundational research in database theory. Consider a scenario in a classroom setting where students are learning to write relational queries and suppose a student has submitted a query  $Q$  for a question they have been asked to solve. Since there are various tools for checking the correctness of the syntax of a query, we can assume that  $Q$  is syntactically correct. Suppose the instructor has a correct query  $Q_0$  as reference for the same question. Since there can be multiple equivalent ways of writing the same query, ideally we want to check whether  $Q$  and  $Q_0$  are equivalent. However, unfortunately, “*query equivalence*” testing is undecidable in general, for non-monotone RC or FOL queries involving universal quantifiers ( $\forall$ ), for *Datalog* with recursion, and for practical query languages such as SQL with support for even just integer arithmetic [24, 25, 1, 5]. The problem is decidable but intractable for non-recursive monotone queries [6, 23], but that does not give a solution for verifying student submissions for general queries they need to learn in practice. “*Eyeballing*” errors manually by instructors and teaching assistants is difficult, especially for subtle mistakes, and does not scale in large classes. Therefore, the standard practice is to run both the student query  $Q$  and the reference query  $Q_0$  on some test database instances  $D$  and compare their results, which is often done by “*autograding*” tools like GradeScope [11]. If the results differ, i.e.,  $Q(D) \neq Q_0(D)$ , we know  $Q$  is wrong. Note that this does not test query equivalence, i.e., there is no guarantee that  $Q$  is correct if the results agree. For practical purposes, we resort to complex test instances that attempt to exercise conceivable corner cases in order to increase the chances of catching wrong queries.

Merely marking  $Q$  as wrong, however, does not guide students toward a correct solution. As a first step, *how do we explain to students “why” their query is wrong?* One option for the instructor to explain the errors in the student query  $Q$  is to show the test database instance  $D$  for which we know  $Q(D) \neq Q_0(D)$  as a “*counterexample*”, together with  $Q(D)$  and  $Q_0(D)$  (without revealing  $Q_0$  itself). This approach may not work since  $D$  tends to be large and complex by design. For example, in the database courses at Duke University, one assignment is based on the real DBLP database with millions of rows; another assignment uses synthetic test databases, and we needed tens of thousands of rows in order to catch most of the errors that were manually found [17]. Showing millions, thousands, or even just dozens of database rows can overwhelm the student, especially when the student is learning to think about the solution logic as well as the query syntax and semantics for the first time. Further, revealing a test instance  $D$  in its entirety encourages the behavior of tweaking one’s query just to pass





■ **Figure 1** Wrong query  $Q$ .

each particular test, which is not conducive to learning. Hence, we need to develop solutions to help students better understand what is wrong, and make it scale for a large number of students and easily customizable for different exercises across classrooms. The problem becomes more challenging when we consider different classes of relational queries, as the solutions for the procedural RA queries and declarative SQL queries may be very different.

At Duke University, we have been working on a project called *HNRQ: Helping Novices Learn and Debug Relational Queries* [27], where we are building a suite of tools for debugging queries. Furthermore, we are using these tools in our large undergraduate and graduate database classes to provide automated and scalable help to students debug their own queries. We are evaluating the effectiveness of these tools on learning by running user studies and surveys employing techniques from CS Education in consultation with the Institutional Review Board (IRB) at our university. These tools are built upon foundations and techniques from database theory, whereas they have simple user-friendly interactive graphical interfaces targeted towards students who are learning to write relational queries. This research direction showcase a practical application of database theory to help students learn to write relational queries and has a direct impact on any data science curriculum. Building query debugging tools has applications beyond the educational setting, e.g., to help debug database queries that fail regression tests commonly used by the software industry.

In the rest of the paper, we give a brief overview of our query debugging tools, and some ongoing and future research directions. We skip discussions of related work in this extended abstract; a detailed discussion of related work can be found in our research papers [17, 10, 16], in the articles in a Data Engineering Bulletin Special Issue on “*Widening the Impact of Data Engineering through Innovations in Education, Interfaces, and Features*” that Roy and Yang co-edited [22], and in the recent workshops “*Data Systems Education (DataEd)*” [21].

### Explaining Wrong Queries with Small Counterexamples

In our first tool called **RATest** [17, 18], we focused on explaining wrong RA queries adapting the idea of using a test instance  $D$  where  $Q(D) \neq Q_0(D)$ . Instead of showing the entire test database instance  $D$ , the key idea behind RATest is to show a small subinstance  $D' \subseteq D$  (still conforming to all database constraints like keys and foreign keys) such that  $Q(D') \neq Q_0(D')$ , i.e.,  $D'$  is a small counterexample still able to illustrate the difference between  $Q$  and  $Q_0$ .

► **Example 1.** Consider the popular Drinker database with information about bars and bar-goers as follows (keys are underlined):

Drinker(name, address), Bar(name, address), Beer(name, brewer),  
 Frequents(drinker, bar, times\_a\_week), Likes(drinker, beer), Serves(bar, beer, price).

name	addr
Eve Edwards	32767 Magic Way

(a) Drinker.

name	brewer
American Pale Ale	Sierra Nevada

(b) Beer.

name	addr
Restaurant Memory	1276 Evans Estate
Tadim	082 Julia Underpass
Restaurante Raffaele	7357 Dalton Walks

(c) Bar.

drinker	beer
Eve Edwards	American Pale Ale

(d) Likes.

bar	beer	price
Restaurant Memory	American Pale Ale	2.25
Restaurante Raffaele	American Pale Ale	2.75
Tadim	American Pale Ale	3.5

(e) Serves.

■ **Figure 2** A small counterexample returned by RATest.

Students are asked to write the following query in RA: “for each beer liked by any drinker with first name **Eve**, find the bars that serve this beer at the highest price.” A common incorrect query  $Q$  is shown in Figure 1 that considers not-lowest price instead of highest price among other errors. The test database instance  $D$  for this database used in our database classes contains thousands of tuples and would not be useful to the student. RATest, on the other hand, is able to find a remarkably small counterexample in Figure 2 automatically to illustrate why  $Q$  is wrong. RATest also shows that  $Q$  returns both  $\langle \text{American Pale Ale}, \text{Restaurante Raffaele} \rangle$  and  $\langle \text{American Pale Ale}, \text{Tadim} \rangle$  on this small instance, whereas the correct result should contain only  $\langle \text{American Pale Ale}, \text{Tadim} \rangle$ . RATest further allows the student to trace the execution of  $Q$  over the counterexample along the RA query plan. The correct query  $Q_0$  itself is never revealed.

In the backend, to simplify the problem of finding a small  $D' \subseteq D$  further for efficiency such that  $Q(D) \neq Q_0(D)$ , we choose a result tuple  $t \in Q_0(D) \setminus Q(D)$  (or  $t \in Q(D) \setminus Q_0(D)$ ), and try to find a small instance  $D' \subseteq D$  such that still  $t \in Q_0(D') \setminus Q(D')$ . If  $Q$  and  $Q_0$  are **monotone**, we can solve this problem efficiently in polynomial time in the size of data (i.e., data complexity [26]). The problem becomes more interesting and challenging when  $Q_0$  or  $Q$  is intrinsically **non-monotone**, then a non-answer can become an answer in a smaller database instance. In [17], we discussed both the data complexity and combined complexity (when both data size and query size are parameters) for different query classes: as soon as queries involve projection, join, and difference operations, even the data complexity becomes NP-hard. Nevertheless, in [17] we provided practical solutions for general RA queries with the difference operation. The intuitive idea is to compute the provenance [13] as a Boolean formula  $\phi = \phi_1 \wedge \neg\phi_2$  for the tuple of interest, say  $t \in (Q_0 - Q)(D)$ , where  $\phi_1, \phi_2$  denote its provenance or lineage [12] in  $Q_0(D), Q(D)$  respectively (joint usage of two tuples by join  $\bowtie$  is captured with  $\wedge$ , and alternative usage by projection  $\pi$  or union  $\cup$  is captured by  $\vee$ ). Then we find a minimum satisfying solution of  $\phi$  by setting the smallest number of variables to true (*min-ones satisfiability problem*), which can be tackled by *SMT (satisfiability modulo theories)* solvers. We use an automatic rewrite procedure to convert the queries into SQL that will compute not only their results but also provenance expressions for the results tuples. We apply optimizations such as pushing down selections for interactive performance.

Suppose Eve likes beer  $B$ ,  
 bar  $X_1$  serves beer  $B$  at price  $P_1$ ,  
 bar  $X_2$  serves beer  $B$  at price  $P_2$ ,  
 bar  $X_3$  serves beer  $B$  at price  $P_3$ ,  
 and  $P_2 < P_1 < P_3$ ;  
 then your query would incorrectly return  $\langle \text{beer } B, \text{bar } X_1 \rangle$ .

■ **Figure 3** Intended generalization of the counterexample in Figure 2.

More challenges arise once we consider extended RA queries with **aggregation, grouping, and HAVING** (i.e., a selection after a group-by aggregation), for which the previous approach may not yield a small counterexample, because its aggregate (say SUM) value generally depends on all member tuples in the input group corresponding to  $t$ , and because of predicates like HAVING Count(\*) > 1000 that seek to output a database with at least 1000 tuples. Our solution is to *parameterize* the queries and allow a counterexample to differentiate two queries on *some* setting of the parameters (which may be different from the original setting) and constructing the provenance for aggregates using the approach by Amsterdamer et al. [4].

We built RATest as a web-based teaching tool, deployed it in an undergraduate database class at Duke university in Fall 2018 with about 170 students, and conducted a detailed *user study* with 10 homework problems where RATest was available only for a subset of the problems. We collected usage patterns on RATest as well as how students eventually scored on the homework problems. The usage of RATest was high in the class and more for harder queries, and RATest seemed to have helped students solve harder queries as well as other similar hard queries. We also collected 134 anonymous responses from the students and the feedback was largely positive. 69.4% of the respondents agreed or strongly agreed that the counterexamples helped them understand or fix the bug in their queries, and 93.2% would like to use similar tools in the future for assignments on querying databases. Open-ended comments were overwhelmingly positive, e.g.,

*“It was incredibly useful debugging edge cases in the larger dataset not provided in our sample dataset with behavior not explicitly described in the problem set.”*

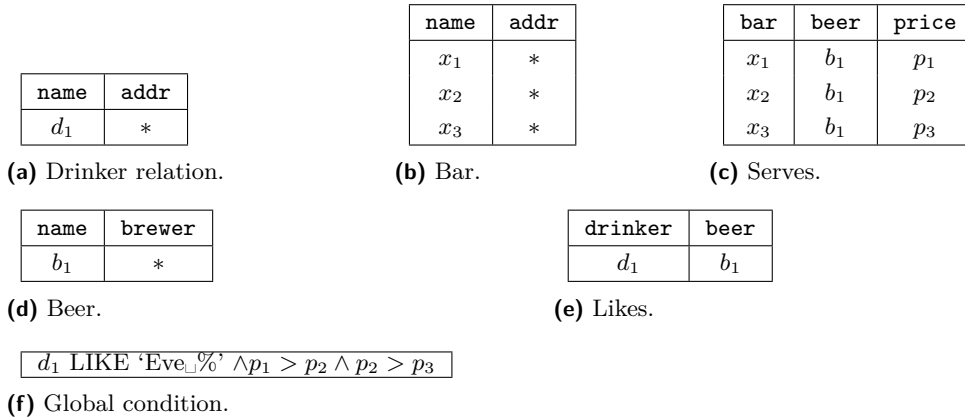
*“Overall, very helpful and would like to see similar testers for future assignments.”*

*“I liked how it gave us a concise example showing what we did wrong.”*

Since Fall 2018, we have used RATest regularly in graduate and undergraduate courses at Duke with continued extensive use and positive feedback from students; till date RATest has been used **by more than 1600 students** at Duke University.

### Explaining Wrong Queries with Abstract Conditional Instances

Since RATest was successful both in terms of research and practical uses in a classroom setting, the next step was improving this tool in terms of usability, generality, and deployability. For instance, in Example 1 and Figure 1, it can be noted that  $Q$  makes multiple mistakes. The counterexample in Figure 2 shows the mistake that for a given beer,  $Q$  actually finds bars that do not serve it at the lowest price (a monotone query), as opposed to bars that serve it at the highest price (requires a non-monotone query). The other mistake, where the predicate “LIKE 'Eve%'” may incorrectly pick up a drinker whose first name is **Evelyn**, is not illustrated in this counterexample. While this omission might be helpful for some students who can focus on one mistake at a time, it is also useful to show all the mistakes in the explanation for why  $Q$  is wrong. Further, Figure 2 does not “pinpoint” the mistake that the error is due to the presence of three distinct price values of the same beer, due to the



■ **Figure 4** An abstract conditional instance generated by CINSGEN generalizing the concrete small instance in Figure 2 and capturing the intuitive explanation in Figure 3.

presence of redundant information in the form of attribute values in several other tuples in other tables. Ideally, we want an explanation as given in Figure 3, which would explain one major error why the query  $Q$  in Figure 1 is wrong. Moreover, the intrinsic limitation of an instance-based explanation starting with a test instance  $D$  is that it may not detect a wrong query. Further, in some cases, a good test instance  $D$  may be unavailable, especially when new queries are created on a new database schema.

Our next tool named **CINSGEN** [10, 15]) focused on addressing these issues. In [10] we considered queries in the form of Relational Calculus (the prototype [15] provided a procedure to convert SQL queries without aggregates to RC). The broad goal in CINSGEN was to understand all possible solutions to a query  $Q'$  by a set of “*generic*” and “*representative*” instances that (1) illustrate different ways the query  $Q'$  can be satisfied, and (2) summarize all specific instances that would satisfy the query in the same way by abstracting away unnecessary details. To formalize this, we develop the concept *conditional instances* or *c-instances* by adapting the notion of *c-tables* by Imilienski and Lipski [14] for incomplete databases, which are abstract database instances comprising variables (labeled nulls) along with a condition on those variables. An example c-instance capturing the intuitive condition in Figure 3 and generalizing the concrete small instance in Figure 2 is shown in Figure 4. Another c-instance (not shown) will illustrate the second error of the first name being “Eve” vs. the first name starting with “Eve”. Thus, each c-instance can be considered a representative of all grounded instances that replace its variables with constants that satisfy the conditions that they are involved in. Since it may be hard to capture all satisfying instances with abstract c-instances (e.g., they can be unbounded in size), we use the idea of *coverage* from the software validation field [19, 20, 3], covering different ways a query can be satisfied. Since now we are essentially testing equivalence of two first order logic queries, the problem of finding such conditional instances in general is **undecidable** by a reduction from the *finite satisfiability problem* and Trakhtenbrot’s Theorem [25]. Hence we developed practical algorithms inspired by the “*chase*” procedure by Fagin et al. [8, 9] with a user-specified input stopping condition (on the number of steps or time) to generate such instances. Hence, if CINSGEN does not return any c-instances, the query  $Q$  may still be wrong and not equivalent to  $Q_0$ . Our user study with undergraduate and graduate students shows that although both RATest [17] and CINSGEN [10] help students detect errors, conditional instances by CINSGEN help students detect multiple errors in wrong queries unlike concrete instances provided by RATest.

### Tracing Outputs and Errors in Declarative SQL Queries

Part of convincing a student why a relational query  $Q$  is wrong is to help the student understand the semantics of  $Q$  and its behavior on an input instance  $D$ . Even if  $D$  is small, it may not be obvious why  $Q$  produces  $Q(D)$  as its result, especially for students who may have misunderstandings about how certain query constructs work. A useful way of debugging  $Q$  would be to “trace” its execution over  $D$ . For RA (or other operator-based relational languages), a straightforward approach, which we have used successfully in [17, 18], is to show the algebraic query expression tree and the intermediate results produced by each operator when executed in a bottom-up fashion. However, for the practical “declarative” language of SQL, which draws inspiration from logic, it is not even clear what “tracing” means. For instance, for a correlated subquery, we cannot talk about its result without the context from which it derives its variable bindings. Using the schema in Example 1, the following simple query has a correlated subquery:

```
SELECT address
FROM Drinker
WHERE EXISTS
(SELECT * FROM Frequenters WHERE drinker = Drinker.name)
```

Clearly, the result of the subquery depends on the particular `Drinker.name` value, which comes from the outer query over `Drinker`. Hence, tracing intermediate results in a bottom-up fashion does not work. In practice, with or without correlated subqueries, SQL queries are almost never executed as the way they were written because of query optimization. Instead, we must be able to trace SQL at a *logical* level, in a way consistent with how a query is originally written, without requiring any additional knowledge of relational algebra or its mapping from SQL. Toward explaining how answers are generated from a SQL query, we developed a tracing tool named **I-REX** [16] that provides a novel interactive interface that allows users to trace query evaluation in a way faithful to how the query is written originally. For instance, I-Rex explains how the results are semantically generated (and non-answers are filtered out) by first computing the cross product of tables in the FROM clause, applying the predicates in the WHERE clause in a logical tree form, grouping intermediate results in the GROUP BY clause, applying predicate in the HAVING clause, and executing the subqueries in a “context” with specific variable bindings provided during the evaluation of its outer queries. In the backend, I-REX extends provenance support for SQL in non-trivial ways to work with various query constructs. This tool is currently being deployed in our classes and evaluations by user studies are being performed.

### Ongoing and Future Work

When a student understands that their query is wrong, the next natural step is to provide some “*hints*” to fix their queries. While it is possible to quickly suggest students to restructure their approaches if the wrong queries are too far off from the solution, many wrong queries contain subtle mistakes which require potentially long time for examining and resolving errors, and instructors need to come up with good hints for fixing errors without showing the solution. Further, for wrong queries with smaller mistakes (e.g., missing a predicate in the WHERE clause), it is important to show the incremental changes that have to be made on the wrong query instead of suggesting a correct query that is drastically different from the student query. Toward this goal, we are developing a tool that takes two non-equivalent SQL queries (a correct query and a wrong query), pinpoints the parts of the wrong query that causes “*semantic*” differences between the queries, and also provides hints for direct

edits to make it equivalent to the correct query. Our approach works for simple single-block SQL queries (without nested sub-queries, NULLs, constraints, etc.), and it will be interesting to develop methods for larger classes of SQL queries (e.g., two non-equivalent queries can become equivalent in the presence of certain integrity constraints).

A much more challenging and open-ended research direction is designing and building query debugging solutions in the realm of Large language Model (LLM)-based tools such as ChatGPT. These tools can return queries in standard relational query languages like SQL, but may produce incorrect solutions for complex queries. While questions like whether and how such tools should be allowed in database classes are being discussed in the community in various panels and workshops, the form of query debugging may be very different in the presence of LLMs, and may lead to a novel direction of research spanning multiple areas such as database theory, natural language processing, and programming languages.

Another important aspect in building query debugging tools is evaluating their effect on learning – e.g., whether they help students learn to write a query when such tools are not available, whether they inspire students to seek help more from instructors, or whether the students merely use them as a shortcut for getting their class assignments completed in less time. While we have been conducting extensive user studies and surveys in the large database classes at our university at different stages of development of these tools, there are restrictions on how such studies can be performed in classroom settings due to several compliance and ethical concerns. For instance, the gold standard of inferring causal conclusions by “*Randomized Controlled Trials*” to test whether these tools help students learn to write queries may not be feasible in an active class. Continued discussions and collaborations across universities will lead to fundamental database research as well as effective scalable solutions potentially revolutionizing database and data science education.

---

## References

- 1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995. URL: <http://webdam.inria.fr/Alice/>.
- 2 Alireza Ahadi, Julia Prior, Vahid Behbood, and Raymond Lister. A quantitative study of the relative difficulty for novices of writing seven different types of sql queries. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE '15*, pages 201–206, New York, NY, USA, 2015. ACM. doi:10.1145/2729094.2742620.
- 3 Paul Ammann and Jeff Offutt. *Introduction to software testing*. Cambridge University Press, 2016.
- 4 Yael Amsterdamer, Daniel Deutch, and Val Tannen. Provenance for aggregate queries. In *PODS*, pages 153–164, 2011. doi:10.1145/1989284.1989302.
- 5 Marcelo Arenas, Pablo Barceló, Leonid Libkin, Wim Martens, and Andreas Pieris. *Database Theory*. Open source at <https://github.com/pdm-book/community>, 2022.
- 6 Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing, STOC '77*, pages 77–90, 1977. doi:10.1145/800105.803397.
- 7 Data Analytics Market Share, Market Research Future. <https://www.marketresearchfuture.com/reports/data-analytics-market-1689>.
- 8 Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: Semantics and query answering. In *ICDT*, pages 207–224, 2003. doi:10.1007/3-540-36285-1\_14.
- 9 Ronald Fagin, Phokion G. Kolaitis, and Lucian Popa. Data exchange: getting to the core. In *PODS*, pages 90–101, 2003. doi:10.1145/773153.773163.
- 10 Amir Gilad, Zhengjie Miao, Sudeepa Roy, and Jun Yang. Understanding queries by conditional instances. In Zachary G. Ives, Angela Bonifati, and Amr El Abbadi, editors, *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12–17, 2022*, pages 355–368. ACM, 2022. doi:10.1145/3514221.3517898.



- 11 Gradescope. <https://www.gradescope.com/>.
- 12 Todd J. Green, Gregory Karvounarakis, and Val Tannen. Provenance semirings. In *PODS*, pages 31–40, 2007. doi:10.1145/1265530.1265535.
- 13 Todd J Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. In *PODS*, pages 31–40, 2007. doi:10.1145/1265530.1265535.
- 14 Tomasz Imielinski and Witold Lipski Jr. Incomplete information in relational databases. *J. ACM*, 31(4):761–791, 1984. doi:10.1145/1634.1886.
- 15 Hanze Meng, Zhengjie Miao, Amir Gilad, Sudeepa Roy, and Jun Yang. Characterizing and verifying queries via CINSGEN. In Sudipto Das, Ippokratis Pandis, K. Selçuk Candan, and Sihem Amer-Yahia, editors, *Companion of the 2023 International Conference on Management of Data, SIGMOD/PODS 2023, Seattle, WA, USA, June 18-23, 2023*, pages 143–146. ACM, 2023. doi:10.1145/3555041.3589721.
- 16 Zhengjie Miao, Tiangang Chen, Alexander Bendeck, Kevin Day, Sudeepa Roy, and Jun Yang. I-rex: An interactive relational query explainer for SQL. *Proc. VLDB Endow.*, 13(12):2997–3000, 2020. doi:10.14778/3415478.3415528.
- 17 Zhengjie Miao, Sudeepa Roy, and Jun Yang. Explaining wrong queries using small examples. In *SIGMOD*, pages 503–520, 2019. doi:10.1145/3299869.3319866.
- 18 Zhengjie Miao, Sudeepa Roy, and Jun Yang. Ratest: Explaining wrong relational queries using small examples. In Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska, editors, *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 1961–1964. ACM, 2019. doi:10.1145/3299869.3320236.
- 19 Joan C. Miller and Clifford J. Maloney. Systematic mistake analysis of digital computer programs. *Commun. ACM*, 6(2):58–63, 1963. doi:10.1145/366246.366248.
- 20 Glenford J Myers, Tom Badgett, Todd M Thomas, and Corey Sandler. *The art of software testing*, volume 2. Wiley, 2004.
- 21 International Workshop on Data Systems Education (DataEd) 2023.
- 22 Sudeepa Roy and Jun Yang. Letter from the special issue editor. *IEEE Data Eng. Bull.*, 45(3):2–3, 2022. URL: <http://sites.computer.org/debull/A22sept/p2.pdf>.
- 23 Yehoshua Sagiv and Mihalis Yannakakis. Equivalences among relational expressions with the union and difference operators. *J. ACM*, 27(4):633–655, oct 1980. doi:10.1145/322217.322221.
- 24 Oded Shmueli. Equivalence of datalog queries is undecidable. *J. Log. Program.*, 15(3):231–241, feb 1993. doi:10.1016/0743-1066(93)90040-N.
- 25 Boris Trakhtenbrot. The impossibility of an algorithm for the decidability problem on finite classes. *Proceedings of the USSR Academy of Sciences*, 70(4):569–572, 1950.
- 26 Moshe Y. Vardi. The complexity of relational query languages (extended abstract). In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, pages 137–146, 1982. doi:10.1145/800070.802186.
- 27 Project website: <https://dukedb-hnrq.github.io/>.





# Rule-Based Ontologies: From Semantics to Syntax

Andreas Pieris  

University of Edinburgh, UK

University of Cyprus, Nicosia, Cyprus

---

## Abstract

An ontology specifies an abstract model of a domain of interest via a formal language that is typically based on logic. Tuple-generating dependencies (tgds) and equality-generating dependencies (egds) originally introduced as a unifying framework for database integrity constraints, and later on used in data exchange and integration, are well suited for modeling ontologies that are intended for data-intensive tasks. The reason is that, unlike other popular formalisms such as description logics, tgds and egds can easily handle higher-arity relations that naturally occur in relational databases. In recent years, there has been an extensive study of tgd- and egd-based ontologies and of their applications to several different data-intensive tasks. In those studies, model theory plays a crucial role and it typically proceeds from syntax to semantics. In other words, the syntax of an ontology language is introduced first and then the properties of the mathematical structures that satisfy ontologies of that language are explored. There is, however, a mature and growing body of research in the reverse direction, i.e., from semantics to syntax. Here, the starting point is a collection of model-theoretic properties and the goal is to determine whether or not these properties characterize some ontology language. Such results are welcome as they pinpoint the expressive power of an ontology language in terms of insightful model-theoretic properties. The main aim of this tutorial is to present a comprehensive overview of model-theoretic characterizations of tgd- and egd-based ontology languages that are encountered in database theory and symbolic artificial intelligence.

**2012 ACM Subject Classification** Theory of computation → Logic

**Keywords and phrases** ontologies, tuple-generating dependencies, equality-generating dependencies, model theory, model-theoretic characterizations

**Digital Object Identifier** 10.4230/LIPICs.ICDT.2024.3

**Category** Invited Talk



© Andreas Pieris;

licensed under Creative Commons License CC-BY 4.0

27th International Conference on Database Theory (ICDT 2024).

Editors: Graham Cormode and Michael Shekelyan; Article No. 3; pp. 3:1–3:1

Leibniz International Proceedings in Informatics




LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany





# Direct Access for Answers to Conjunctive Queries with Aggregation

Idan Eldar  

Technion – Israel Institute of Technology, Haifa, Israel

Nofar Carmeli  

Inria, LIRMM, Univ Montpellier, CNRS, France

Benny Kimelfeld  

Technion – Israel Institute of Technology, Haifa, Israel

---

## Abstract

We study the fine-grained complexity of conjunctive queries with grouping and aggregation. For some common aggregate functions (e.g., min, max, count, sum), such a query can be phrased as an ordinary conjunctive query over a database annotated with a suitable commutative semiring. Specifically, we investigate the ability to evaluate such queries by constructing in log-linear time a data structure that provides logarithmic-time direct access to the answers ordered by a given lexicographic order. This task is nontrivial since the number of answers might be larger than log-linear in the size of the input, and so, the data structure needs to provide a compact representation of the space of answers.

In the absence of aggregation and annotation, past research provides a sufficient tractability condition on queries and orders. For queries without self-joins, this condition is not just sufficient, but also necessary (under conventional lower-bound assumptions in fine-grained complexity). We show that all past results continue to hold for annotated databases, assuming that the annotation itself is not part of the lexicographic order. On the other hand, we show infeasibility for the case of count-distinct that does not have any efficient representation as a commutative semiring. We then investigate the ability to include the aggregate and annotation outcome in the lexicographic order. Among the hardness results, standing out as tractable is the case of a semiring with an idempotent addition, such as those of min and max. Notably, this case captures also count-distinct over a logarithmic-size domain.

**2012 ACM Subject Classification** Theory of computation → Database query languages (principles); Theory of computation → Database query processing and optimization (theory)

**Keywords and phrases** aggregate queries, conjunctive queries, provenance semirings, commutative semirings, annotated databases, direct access, ranking function, answer orderings, query classification

**Digital Object Identifier** 10.4230/LIPIcs.ICDT.2024.4

**Related Version** *Full Version*: <https://arxiv.org/abs/2303.05327>

**Funding** The work of Idan Eldar and Benny Kimelfeld was supported by the German Research Foundation (DFG) grant KI 2348/1-1 (DIP Program).

## 1 Introduction

Consider a query  $Q$  that may have a large number of answers, say cubic in the number of tuples of the input database  $D$ . By answering  $Q$  via *direct access*, we avoid the materialization of the list of answers, and instead, construct a compact data structure  $S$  that supports random access: given an index  $i$ , retrieve the  $i$ th answer. Hence, direct access evaluation for a query  $Q$  consists of two algorithms, one for the structure construction (with the input  $D$ ), called *preprocessing*, and one for fast access to the answers (with the input  $S$  and  $i$ ). This task is nontrivial when  $S$  is considerably cheaper to construct than  $Q(D)$ . Similarly to past work on direct access [6], we adopt the tractability requirement of linear or quasi-linear time



© Idan Eldar, Nofar Carmeli, and Benny Kimelfeld;  
licensed under Creative Commons License CC-BY 4.0  
27th International Conference on Database Theory (ICDT 2024).

Editors: Graham Cormode and Michael Shekelyan; Article No. 4; pp. 4:1–4:20

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

to construct  $S$ , and logarithmic time per access. Hence, up to a poly-logarithmic factor, the required construction time is what it takes to read the database (i.e., linear time), and the access time is constant. The structure  $S$  can be viewed as a compact representation of  $Q(D)$ , in the general sense of Factorized Databases [17], since its size is necessarily quasi-linear and it provides fast access.

Direct access solutions have been devised for Conjunctive Queries (CQs), first as a way to establish algorithms for enumerating the answers with linear preprocessing time and constant delay [4] (and FO queries with restrictions on the database [2]); the preprocessing phase constructs  $S$ , and the enumeration phase retrieves the answers by accessing  $S$  with increasing indices  $i$ . Later, direct access had a more crucial role within the task of enumerating the answers in a uniformly random order [7]. As a notion of query evaluation, direct access is interesting in its own right, since we can view  $S$  itself as the “result” of the query in the case where array-like access is sufficient for downstream processing (e.g., to produce a sample of answers, to return answers by pages, to answer  $q$ -quantile queries, etc.). But then  $S$  has the benefit that it is considerably smaller and faster to produce than the materialized set of answers. Indeed, recent work has studied the complexity of direct access independently (regardless of any enumeration context) [5], and specifically studied which *orders* over the answers allow for such evaluation [6]. In this paper, we continue with the line of work by Carmeli et al. [6] and investigate the ability to support query evaluation via direct access for *aggregate queries*, while focusing on lexicographic orderings of answers.

For illustration, consider the following example, inspired by the FIFA World Cup. Suppose that we have a database of players of teams (countries), sponsors of teams, and goals scored in different games. Specifically, we have three relations:  $\text{TEAMS}(\text{player}, \text{country})$ ,  $\text{SPONSORS}(\text{org}, \text{country})$ , and  $\text{GOALS}(\text{game}, \text{player}, \text{time})$ . The following CQ finds times when sponsors got exposure due to goals of supported teams:

$$Q_1(c, o, p, t) :- \text{TEAMS}(p, c), \text{SPONSORS}(o, c), \text{GOALS}(g, p, t)$$

Suppose also that we would like the answers to be ordered lexicographically by their order in the head: first by  $c$  (country), then by  $o$  (organization), then by  $p$  (player), and lastly by  $t$  (time). Note that  $o$ ,  $c$ ,  $p$  and  $t$  are *free* variables and  $g$  is an *existential* variable. Carmeli et al. [6] studied the ability to evaluate such ordered queries with direct access. In the case of  $Q_1$ , the results of Carmeli et al. show that there is an efficient direct access evaluation (since the query is free-connex and there is no “disruptive trio”).

Now, suppose that we would like to count the goals per sponsorship and player. In standard CQ notation (e.g., Cohen et al. [9]), we can phrase this query as follows.

$$Q_2(c, o, p, \text{Count}()) :- \text{TEAMS}(p, c), \text{SPONSORS}(o, c), \text{GOALS}(g, p, t)$$

Here, the variables  $p$ ,  $c$ , and  $o$  are treated as the *grouping variables* (rather than free variables), where each combination of values defines a group of tuples over  $(c, o, p, g, t)$  and  $\text{Count}()$  simply counts the tuples in the group. Again, we would like to answer this query via direct access. This introduces two challenges. The first challenge is *aggregate construction*: when we access a tuple using  $S$ , the aggregate value should be quickly produced as well. The second challenge is *ordering by aggregation*: how can we incorporate the aggregation in the lexicographic order of the answers if so desired by the query? As an example, we may wish to order the answers first by  $c$ , then by  $\text{Count}()$ , and then by  $o$  and  $p$ ; in this case, we would phrase the head accordingly as  $Q_2(c, \text{Count}(), o, p)$ .

As previously done in the context of algorithms for aggregate queries [15, 20], we also study a semiring alternative to the above formalism of aggregate queries. Specifically, we can adopt the well-known framework of *provenance semiring* of Green, Karvounarakis and

Tannen [13] and phrase the query as an ordinary CQ with the annotation carrying the aggregate value (e.g., the number of goals in our example). To reason about random-access evaluation, we found it more elegant, general, and insightful to support CQs over annotated databases rather than SQL-like aggregate functions. For illustration, we can phrase the above aggregate query  $Q_2$  as the following CQ  $Q_3$ , but for a database that is annotated using a specific commutative semiring.

$$Q_3(c, o, p) :- \text{TEAMS}(p, c), \text{SPONSORS}(o, c), \text{GOALS}(g, p, t)$$

In a nutshell (the formal definition is in Section 2), the idea is that each tuple is annotated with an element of the semiring, the annotation of each tuple in the group is the product of the participating tuple annotations, and the annotation of the whole group is the sum of all tuple annotations in the group's tuples. In the case of our example with  $Q_3$ , we use the numeric semiring  $(\mathbb{Z}, +, \cdot, 0, 1)$ , and each tuple is annotated simply with the number 1. We can use different semirings and annotations to compute different aggregate functions like sum, min, and max. Here again, we have challenges analogous to the aggregate case: *annotation construction* and *ordering by annotation*. The previous example becomes ordering by  $c$ , then by the annotation, and then by  $o$  and  $p$ . Notationally, we specify the annotation position by the symbol  $\star$  and phrase the query as  $Q_3(c, \star, o, p) :- \text{TEAMS}(p, c), \text{SPONSORS}(o, c), \text{GOALS}(g, p, t)$ . We refer to such a query as a CQ $^\star$ .

In this paper, we study queries in both formalisms – CQs enhanced with aggregate functions and ordinary CQ $^\star$ s over annotated databases. We usually devise algorithms and upper bounds on general commutative semirings (possibly with additional conditions), as positive results carry over to the aggregate formalism, and we prove cases of specific intractable queries with specific aggregate functions over ordinary (non-annotated) databases.

Our analysis is done in two parts. In Section 4, we study the case where the annotation or aggregation is *not* a part of the lexicographic order; we show that under reasonable assumptions about the complexity of the semiring operations, all results for ordinary databases [6] continue to hold in the presence of annotation (that is, we can solve annotation construction). We conclude the analogous tractability for the common aggregate functions (count, sum, min, max, average), with the exception of count-distinct which cannot be expressed efficiently as a semiring annotation. In Section 5, we study the ability to include the annotation or aggregation in nontrivial positions within the lexicographic order; we show that the picture is more involved there since we hit hardness very quickly, and we need to carefully state the conditions that allow for efficient direct access for important cases.

The remainder of the paper is organized as follows. After preliminary concepts and notation in Section 2, Section 3 defines the challenge of direct access with an underlying order and recalls the state of affairs for ordinary CQs over ordinary databases. We present our analysis in Sections 4 and 5 (as explained in the previous paragraph), and conclude in Section 6. Missing proofs can be found in the full version of the paper [10].

## 2 Preliminaries

We begin with preliminary notation and terminology that we use throughout the paper.

**Databases and conjunctive queries.** A *schema*  $\mathbf{S}$  is a finite set  $\{R_1, \dots, R_k\}$  of relation symbols. Each relation symbol  $R$  is associated with an arity  $\text{ar}(R)$ , which is a natural number. We assume a countably infinite set  $\text{Const}$  of *constants* that appear as values of databases. A *database*  $D$  over a schema  $\mathbf{S}$  maps every relation symbol  $R$  of  $\mathbf{S}$  to a finite relation  $R^D \subseteq \text{Const}^{\text{ar}(R)}$ . If  $(c_1, \dots, c_k)$  is a tuple of  $R^D$  (where  $k = \text{ar}(R)$ ), then we call the expression  $R(c_1, \dots, c_k)$  a *fact* of  $D$ .

#### 4:4 Direct Access for Answers to Aggregate Queries

A *Conjunctive Query (CQ)* over the schema  $\mathbf{S}$  has the form  $Q(\vec{x}) :- \varphi_1(\vec{x}, \vec{y}), \dots, \varphi_\ell(\vec{x}, \vec{y})$  where  $\vec{x}$  and  $\vec{y}$  are disjoint sequences of variables, and each  $\varphi_i(\vec{x}, \vec{y})$  is an *atomic query*  $R(z_1, \dots, z_k)$  such that  $R \in \mathbf{S}$  with  $\text{ar}(R) = k$  and each  $z_i$  is a variable in  $\vec{x}$  or  $\vec{y}$ . Each  $\varphi_i(\vec{x}, \vec{y})$  is an *atom* of  $Q$ , and we denote by  $\text{atoms}(Q)$  the set of atoms of  $Q$ . We call  $Q(\vec{x})$  the *head* of the query and  $\varphi_1(\vec{x}, \vec{y}), \dots, \varphi_\ell(\vec{x}, \vec{y})$  the *body* of the query. The variables of  $\vec{x}$  are the *free* variables of  $Q$ , and those of  $\vec{y}$  are the *existential* variables of  $Q$ , and every variable occurs at least once in the body. We use  $\text{vars}(Q)$  and  $\text{free}(Q)$  to denote the set of all variables and all free variables of  $Q$ , respectively. If  $\varphi \in \text{atoms}(Q)$ , then  $\text{vars}(\varphi)$  is the set of variables in  $\varphi$ . We say that  $Q$  is *full* if it has no existential variables, that is  $\text{vars}(Q) = \text{free}(Q)$ .

We refer to a database  $D$  over the schema  $\mathbf{S}$  of the CQ  $Q$  as a *database over  $Q$* . A *homomorphism* from a CQ  $Q$  to a database  $D$  over  $Q$  is a mapping  $h$  from the variables of  $Q$  into values of  $D$  such that for each atom  $R(z_1, \dots, z_k)$  of  $Q$  it holds that  $R(h(z_1), \dots, h(z_k))$  is a fact of  $D$ . We denote by  $\text{Hom}(Q, D)$  the set of all homomorphisms from  $Q$  to  $D$ . If  $h \in \text{Hom}(Q, D)$  then we denote by  $h(\vec{x})$  the tuple obtained by replacing every variable  $x$  with the constant  $h(x)$ , and we denote by  $h(\varphi_i(\vec{x}, \vec{y}))$  the fact that is obtained from the atom  $\varphi_i(\vec{x}, \vec{y})$  by replacing every variable  $z$  with the constant  $h(z)$ . An *answer* to  $Q$  over  $D$  is a tuple of the form  $h(\vec{x})$  where  $h \in \text{Hom}(Q, D)$ . The *result* of  $Q$  over  $D$ , denoted  $Q(D)$ , is  $Q(D) := \{h(\vec{x}) \mid h \in \text{Hom}(Q, D)\}$ .

Consider a CQ  $Q(\vec{x}) :- \varphi_1(\vec{x}, \vec{y}), \dots, \varphi_\ell(\vec{x}, \vec{y})$ . We may refer to  $Q$  as  $Q(\vec{x})$  to specify the sequence of free variables in the head. In this work, the *order* of the sequence  $\vec{x}$  has a crucial role, since it determines the desired order of answers. Specifically, we will assume that the desired order of answers is *lexicographic* in the left-to-right order of  $\vec{x}$ . For example, the CQ  $Q(x_1, x_2) :- R(x_1, x_2), S(x_2, y)$  differs from the CQ  $Q'(x_2, x_1) :- R(x_1, x_2), S(x_2, y)$  not only in the order of values within each answer tuple but also in the order over the answers. For  $Q(x_1, x_2)$  we order the answers first by  $x_1$  and then by  $x_2$ , and for  $Q'(x_2, x_1)$  we order first by  $x_2$  and then by  $x_1$ ,

As usual, we associate a CQ  $Q$  with the hypergraph  $H(Q) = (V_Q, E_Q)$  where  $V_Q = \text{vars}(Q)$  and  $E_Q = \{\text{vars}(\varphi) \mid \varphi \in \text{atoms}(Q)\}$ . We say that  $Q$  is *acyclic* if  $H(Q)$  is an ( $\alpha$ -)acyclic hypergraph. Recall that a hypergraph  $H = (V, E)$  is acyclic if there is a tree  $T = (E, E_T)$ , called a *join tree* of  $H$ , with the running intersection property: for each vertex  $v \in V$ , the set of hyperedges that contain  $v$  induces a connected subtree of  $T$ . If  $H$  is acyclic and  $S \subseteq V$ , then we say that  $H$  is *S-connex* if  $H$  remains acyclic even if we add  $S$  to the set of hyperedges [4]. An acyclic CQ  $Q$  is *free-connex* if  $H(Q)$  is acyclic and  $\text{free}(Q)$ -connex.

A hypergraph  $H' = (V, E')$  is an *inclusive extension* of a hypergraph  $H = (V, E)$  if  $E \subseteq E'$  and for every edge  $e' \in E'$  there is an edge  $e \in E$  such that  $e' \subseteq e$ . It is known that  $H$  is acyclic  $S$ -connex if and only if  $H$  has an inclusive extension with a join tree  $T$  such that  $S$  is precisely the set of all variables contained in the vertices of some subtree of  $T$  [1]. We call such a tree *ext-S-connex tree*. When  $S$  is the set of free variables of the CQ, and the CQ is clear from the context, we call such a tree *ext-free-connex*.

The notion of a *disruptive trio* has been introduced previously in the context of direct access to the answers of CQs [6]. A disruptive trio of a CQ  $Q(\vec{x})$  is a set of three distinct free variables  $x_1, x_2$ , and  $x_3$  such that  $x_1$  and  $x_2$  neighbor  $x_3$  but not each other, and  $x_3$  succeeds both  $x_1$  and  $x_2$  in  $\vec{x}$ . By saying that  $x$  and  $y$  are *neighbors* we mean that they occur jointly in at least one of the atoms.

**Aggregate queries.** By an *aggregate function* we refer to a function that takes as input a bag of tuples over  $\text{Const}$  and returns a single value in  $\text{Const}$ . We adopt the notation of Cohen et al. [9] to our setting, as follows. An *aggregate query* here is an expression of the form

$$Q(\vec{x}, \alpha(\vec{w}), \vec{z}) :- \varphi_1(\vec{x}, \vec{y}, \vec{z}), \dots, \varphi_\ell(\vec{x}, \vec{y}, \vec{z})$$

such that  $Q'(\vec{x}, \vec{z}) := \varphi_1(\vec{x}, \vec{y}, \vec{z}), \dots, \varphi_\ell(\vec{x}, \vec{y}, \vec{z})$  is a CQ,  $\alpha$  an aggregate function, and  $\vec{w}$  a sequence of variables from  $\vec{y}$ . An example is  $Q(x_1, x_2, \text{Sum}(y_2), z) := R(x_1, x_2, y_1), S(y_1, y_2, z)$ . We refer to such a query as an *Aggregate CQ* or *ACQ* for short. Given a database  $D$  over  $Q'$ , the result  $Q(D)$  is defined by  $Q(D) := \{(\vec{a}, \alpha(B(\vec{a}, \vec{b})), \vec{b}) \mid (\vec{a}, \vec{b}) \in Q'(D)\}$  where  $B(\vec{a}, \vec{b})$  is the bag that is obtained by collecting the tuples  $h(\vec{w})$  from every  $h \in \text{Hom}(Q', D)$  with  $h(\vec{x}) = \vec{a}$  and  $h(\vec{z}) = \vec{b}$ . Note that our database and query model use set semantics, and we use bag semantics only to define the aggregate functions (in order to capture important functions such as count and sum).

We say that  $Q$  is *acyclic* if  $Q'$  is acyclic. Similarly,  $Q$  is *free-connex* if  $Q'$  is free-connex. A *disruptive trio* of  $Q$  is a disruptive trio of  $Q'$ ; in other words, the definition of a disruptive trio remains unchanged when introducing aggregates, while we consider only the grouping variables and not the aggregate function.

► **Remark 1.** We remark on two aspects in our definition of ACQs. First, the reason for using both  $\vec{x}$  and  $\vec{z}$  as sequences of free variables is to determine a position for aggregate value  $\alpha(\vec{w})$  and, consequently, define its position in the lexicographic order over the answers. Second, the reader should note that, in our notation, an ACQ has a single aggregate function. While this is important for some of our results, other results can be easily extended to multiple aggregate functions  $\alpha(\vec{w}_1), \dots, \alpha(\vec{w}_k)$ . We will mention this extension when relevant. ◻

In this work, we restrict the discussion to the common aggregate functions **Count**, **CountD** (count distinct), **Sum**, **Avg** (average), **Min** and **Max**. All aggregate functions take a single column as input (i.e.,  $\vec{y}_i$  is of length one) except for **Count** that counts the tuples in the group and takes no argument. For instance, the query  $Q_2$  in the Introduction uses **Count**() and it could also use **CountD**( $g$ ) for counting the distinct games with scored goals.

**Commutative semirings.** A *commutative monoid* is an algebraic structure  $(\mathbb{K}, \cdot)$  over a domain  $\mathbb{K}$ , with a binary operation  $\cdot$  that satisfies *associativity*:  $(a \cdot b) \cdot c = a \cdot (b \cdot c)$  for any  $a, b, c \in \mathbb{K}$ , *commutativity*:  $a \cdot b = b \cdot a$  for any  $a, b \in \mathbb{K}$ , and *identity element*: there exists an element  $\emptyset \in \mathbb{K}$  such that  $a \cdot \emptyset = a$  for any  $a \in \mathbb{K}$ . A *commutative semiring* is an algebraic structure  $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$  over a domain  $\mathbb{K}$ , with two binary operations  $\oplus$  and  $\otimes$  and two distinguished elements  $\bar{0}$  and  $\bar{1}$  in  $\mathbb{K}$  that satisfy the following conditions: (a)  $(\mathbb{K}, \oplus)$  is a commutative monoid with the identity element  $\bar{0}$ ; (b)  $(\mathbb{K}, \otimes)$  is a commutative monoid with the identity element  $\bar{1}$ ; (c)  $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$  for all  $a, b, c \in \mathbb{K}$ ; and (d)  $a \otimes \bar{0} = \bar{0}$  for all  $a \in \mathbb{K}$ .

We refer to  $\oplus$  as the *addition* operation,  $\otimes$  as the *multiplication* operation,  $\bar{0}$  as the *additive identity* and  $\bar{1}$  as the *multiplicative identity*. We give examples of commutative semirings at the end of this section.

**Annotated databases and query answers.** Let  $\mathbf{S}$  be a schema and  $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$  a commutative semiring. A  $\mathbb{K}$ -*database* (over  $\mathbf{S}$ ) is a pair  $(D, \tau)$  where  $D$  is a database over  $\mathbf{S}$  and  $\tau : D \rightarrow \mathbb{K}$  is function that maps every fact  $f$  of  $D$  to an element  $\tau(f)$  of  $\mathbb{K}$ , called the *annotation* of  $f$ .

The annotation of a database propagates to the query answers by associating a semiring operation with each algebraic operation [13]. In the case of CQs, the relevant operations are *joins* and *projection*. For join, the annotation of the result is the product of the annotation of the operands. For projection, the annotation is the sum of the annotations of the tuples that give rise to the result. In our terminology, we have the following.

Let  $Q(\vec{x}) := \varphi_1(\vec{x}, \vec{y}), \dots, \varphi_\ell(\vec{x}, \vec{y})$  be a CQ and  $(D, \tau)$  an annotated database. For a homomorphism  $h$  from  $Q$  to  $D$ , we denote by  $\otimes h$  the product of the annotations of the facts in the range of  $h$ , that is  $\otimes h := \tau(h(\varphi_1(\vec{x}, \vec{y}))) \otimes \dots \otimes \tau(h(\varphi_\ell(\vec{x}, \vec{y})))$ . An *answer* to  $Q$  over  $(D, \tau)$  is a pair  $(\vec{c}, a)$  such that  $\vec{c} \in Q(D)$  and

$$a = \oplus \{ \otimes h \mid h \in \text{Hom}(Q, D) \wedge h(\vec{x}) = \vec{c} \}$$

where, for  $A = \{a_1, \dots, a_n\} \subseteq \mathbb{K}$ , we define  $\oplus A = a_1 \oplus \dots \oplus a_n$ . As before, the *result* of  $Q$  over  $(D, \tau)$ , denoted  $Q(D, \tau)$ , is the set of answers  $(\vec{c}, a)$  to  $Q$  over  $(D, \tau)$ . We will make use of the fact that, over commutative semirings, projections and joins are commutative [13].

In this work, we study the ability to incorporate the annotation in the order over the answers. More precisely, we will investigate the complexity of involving the annotation in the lexicographic order over the answers, as if it were another value in the tuple. So, when we consider a CQ  $Q(\vec{x})$ , we need to specify where the annotation goes inside  $\vec{x}$ . Similarly to the ACQ notation, we do so by replacing  $\vec{x}$  with a sequence  $(\vec{x}, \star, \vec{z})$  where  $\star$  represents the annotation value. We refer to a CQ of this form as a CQ $^*$ . An example of a CQ $^*$  is  $Q(x_1, x_2, \star, z) := R(x_1, x_2, y_1), S(y_1, y_2, z)$  where the lexicographic order is by  $x_1$ , then by  $x_2$ , then by the annotation, and then by  $z$ .

Let  $Q$  be a CQ $^*$ , and let  $Q'$  be the CQ obtained from  $Q$  by removing  $\star$  from the head. As in the case of ACQs,  $Q$  is *acyclic* if  $Q'$  is acyclic,  $Q$  is *free-connex* if  $Q'$  is free-connex, and a *disruptive trio* of  $Q$  is a disruptive trio of  $Q'$ .

Aggregate functions can often be captured by annotations of answers in annotated databases, where each aggregate function might require a different commutative semiring:

- **Sum:** the numeric semiring  $(\mathbb{Q}, +, \cdot, 0, 1)$ .
- **Count:** the counting semiring  $(\mathbb{N}, +, \cdot, 0, 1)$ .
- **Max:** the max tropical semiring  $(\mathbb{Q} \cup \{-\infty\}, \max, +, -\infty, 0)$ .
- **Min:** the min tropical semiring  $(\mathbb{Q} \cup \{\infty\}, \min, +, \infty, 0)$ .

The translation is straightforward (and known, e.g., [15, 20]), as we illustrate in Figure 1: the aggregated value becomes the annotation on one of the relations, the annotation outside of this relation is the multiplicative identity (as we later term “locally annotated”), and the addition operation captures the aggregate function. Note that in the case of the numeric and min/max tropical semirings, we are using the domain  $\mathbb{Q}$  of rational numbers rather than all real numbers to avoid issues of numerical presentation in the computational model.

Avg can be computed using Sum and Count. CountD (count distinct) cannot be captured by a semiring, as the result of  $\oplus$  cannot be computed from two intermediary annotations in the domain. We can, however, capture a semantically similar concept with the set semiring  $(\mathcal{P}(\Omega), \cup, \cap, \emptyset, \Omega)$  by annotating each fact with the actual set of distinct elements. However, in such cases, we will need our complexity analysis to be aware of the cost of the operations.

### 3 The Direct-Access Problem

In this paper, we study CQs with lexicographic orders over the answers. As said earlier, the lexicographic order for the CQ  $Q(\vec{x})$  is left to right according to  $\vec{x}$ . We will also investigate lexicographic orders that involve the annotation or aggregation when the query is a CQ $^*$   $Q(\vec{x}, \star, \vec{z})$  or an ACQ  $Q(\vec{x}, \alpha(\vec{u}), \vec{z})$ , respectively. We refer uniformly to the annotation of an answer (over an annotated database) and to the aggregate value of the answer’s group (over an ordinary database) as the *computed value*.

Let  $Q$  be a CQ, CQ $^*$ , or an ACQ. A *direct access* solution for  $Q$  consists of two algorithms: one for *preprocessing* and one for *access*.



TEAMS		GOALS			REPLAYS			TEAM			GOALS				REPLAYS		
$p$	$c$	$g$	$p$	$t$	$g$	$t$	$\Rightarrow$	$p$	$c$	$\tau_+$	$g$	$p$	$t$	$\tau_+$	$g$	$t$	$\tau_+$
1	5	1	1	31	1	1		1	5	1	1	1	31	1	1	1	1
2	5	1	3	50	1	31		2	5	1	1	3	50	1	1	31	31
3	6	1	3	75	1	50		3	6	1	1	3	75	1	1	50	50
4	7	2	4	90	2	5		4	7	1	2	4	90	1	2	5	5
5	8	2	4	9	1	90		5	8	1	2	4	9	1	1	90	90

■ **Figure 1** An example of a  $\mathbb{Q}$ -database over the numerical semiring constructed to evaluate the ACQ  $Q(c, \text{Sum}(t)) :- \text{TEAMS}(p, c), \text{GOALS}(g, p, t), \text{REPLAYS}(g, t)$ .

- The preprocessing algorithm takes as input a database  $D$  over  $Q$  and constructs a data structure  $S_D$ .
- The access algorithm takes as input  $S_D$  and an index  $i$ , and returns the  $i$ th answer of  $Q(D)$  in the lexicographic order. Note that this answer includes the computed value, when it exists. If  $i > |Q(D)|$  then the algorithm should return *null*.

To define the complexity requirements of *efficient* direct access, we first describe the complexity model that we adopt. We use *data complexity* as a yardstick of tractability. Hence, complexity is measured in terms of the size of the database, while the size of the query is fixed (and every query is a separate computational problem). Assuming the input is of size  $n$ , we use the RAM model of computation with  $O(\log n)$ -bit words and uniform-cost operations. Notably, this model allows us to assume perfect hash tables can be constructed in linear time, and they provide access in constant time [12].

Let  $T_p$  and  $T_a$  be numeric functions. A direct-access algorithm is said to be in  $\langle T_p, T_a \rangle$  if the preprocessing phase takes  $O(T_p(|D|))$  time and each access takes  $O(T_a(|D|))$  time. For example,  $\langle \text{loglinear}, \log \rangle$  states that preprocessing constructs in  $O(|D| \log |D|)$  time a data structure that provides  $O(\log |D|)$ -time access. In this work, a query  $Q$  has *efficient* direct access (and  $Q$  is deemed tractable) if it has a direct access algorithm in  $\langle \text{loglinear}, \log \rangle$ .

Carmeli et al. [6] established a dichotomy in the tractability of the CQs and lexicographic orders. This dichotomy relies on the following hypotheses.

- **SparseBMM**: two binary matrices  $A^{n \times n}$  and  $B^{n \times n}$  represented by lists of their non-zero entries cannot be multiplied in  $O(m \text{ polylog}(m))$  time where  $m$  is the total number of non-zero entries in  $A$ ,  $B$  and  $A \times B$ .
- **HYPERCLIQUE**: for all  $k \geq 2$ , there is no  $O(m \text{ polylog}(m))$ -time algorithm that, given a hypergraph with  $m$  hyperedges, determines whether there exists a set of  $k + 1$  vertices such that every subset of  $k$  vertices among them forms a hyperedge.

► **Theorem 2** ([6]). *Let  $Q$  be a CQ.*

1. *If  $Q$  is free-connex with no disruptive trio, then direct access for  $Q$  is in  $\langle \text{loglinear}, \log \rangle$ .*
2. *Otherwise, if  $Q$  is also self-join-free, then direct access for  $Q$  is not in  $\langle \text{loglinear}, \log \rangle$ , assuming the **HYPERCLIQUE** hypothesis (in case  $Q$  is cyclic) and the **SparseBMM** hypothesis (in case  $Q$  is acyclic).*

## 4 Incorporating Annotation and Aggregation in the Answers

In this section, we discuss the existence of efficient direct access in the case where the order *does not* involve the computed value, that is, the annotation (for CQ\*s) or the aggregate values (for ACQs). Equivalently, these are queries where the computed value is last in order,

that is, the vector  $\vec{z}$  in the head is empty. Hence, we focus on CQ\*s of the form  $Q(\vec{x}, \star)$  and ACQs of the form  $Q(\vec{x}, \alpha(\vec{w}))$ . In other words, the problem is similar to the CQ case, except that the access algorithm should also retrieve the aggregated value from the data structure. In Section 4.1, we will use annotated databases to identify the cases where this can be done efficiently for min, max, count, sum, and average. By contrast, in Section 4.2 we will show that we cannot do the same for count-distinct, even in the case of an extremely simple query, unless the domain of the elements we count is small (logarithmic-size).

But first, we need to be clear about the complexity of the semiring operations. The RAM model allows us to assume that the numeric, counting, min tropical, and max tropical semirings use constant space for representing values and constant time for the operations  $\oplus$  and  $\otimes$ . In fact, it suffices for our results to assume that the operations take logarithmic time, and later we will make use of this relaxed assumption (within a special case of CountD). We refer to a semiring with this property as a *logarithmic-time (commutative) semiring*.

In our proofs, we will use the following definition of when two facts  $f$  and  $f'$  agree in the context of two queries. Intuitively, facts agree if they assign the same variables with the same values.

► **Definition 3.** Let  $Q$  and  $Q'$  be CQs over the schemas  $\mathbf{S}$  and  $\mathbf{S}'$ , respectively. Let  $\varphi$  and  $\varphi'$  be atoms of  $Q$  and  $Q'$  over the relation symbols  $R$  and  $R'$ , respectively. Let  $f$  and  $f'$  be facts over  $R$  and  $R'$ , respectively. We say that  $f$  and  $f'$  agree (w.r.t.  $\varphi$  and  $\varphi'$ ) if there exists a homomorphism  $h : \text{vars}(\varphi) \cup \text{vars}(\varphi') \rightarrow \text{Const}$  such that  $f$  and  $f'$  are obtained from  $\varphi$  and  $\varphi'$ , respectively, by replacing each variable  $x$  with the constant  $h(x)$ .

## 4.1 Generalized Dichotomies

We now show that Theorem 2 extends to databases with annotations, and so, also to queries with aggregate functions that can be efficiently simulated by annotations. The first step is to eliminate the existential variables from the CQ\*. It is folklore that free-connex CQs (over non-annotated databases) can be transformed into *full* acyclic CQs in linear time [14, 18]. The following lemma states that the same holds for free-connex CQ\*s over annotated databases.

► **Lemma 4.** Let  $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$  be a logarithmic-time commutative semiring, and let  $Q(\vec{x}, \star, \vec{z})$  be a free-connex CQ\*. There exists a full acyclic CQ\*  $Q'(\vec{x}, \star, \vec{z})$ , without self-joins, and an  $O(|D| \log |D|)$ -time algorithm that maps  $\mathbb{K}$ -databases  $(D, \tau)$  of  $Q$  to  $\mathbb{K}$ -databases  $(D', \tau')$  of  $Q'$  such that (a)  $Q'(D', \tau') = Q(D, \tau)$ ; (b) the variables in every atom of  $Q'$  are contained in an atom of  $Q$ ; and (c)  $Q$  has a disruptive trio if and only if  $Q'$  has a disruptive trio.

We note that in the case where the semiring operations require only constant time, the algorithm of Lemma 4 runs in linear time instead of loglinear time. With this lemma, we can now prove the following generalization of Theorem 2 to annotated databases.

► **Theorem 5.** Let  $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$  be a logarithmic-time commutative semiring, and let  $Q(\vec{x}, \star)$  be a CQ\*.

1. If  $Q$  is free-connex and with no disruptive trio, then direct access for  $Q$  is in  $\langle \text{loglinear}, \log \rangle$  on  $\mathbb{K}$ -databases.
2. Otherwise, if  $Q$  is also self-join-free, then direct access for  $Q$  is not in  $\langle \text{loglinear}, \log \rangle$ , assuming the *HYPERCLIQUE* hypothesis (in case  $Q$  is cyclic) and the *SparseBMM* hypothesis (in case  $Q$  is acyclic).

**Proof.** For the negative side of the dichotomy, we simply use the negative side of Theorem 2. This can be done since each answer to a CQ\* contains the ordinary (non-annotated) answer to the CQ obtained by removing  $\star$ , and the answers have the same order. It is left to prove the positive side of the dichotomy.

We can use Lemma 4 to focus on full CQ\*s without self-joins. We build on the algorithm that Carmeli et al. [6] presented for proving what we gave here as Theorem 2. This algorithm uses the concept of a *layered join tree*, defined for a CQ  $Q$ . It can be seen as a particular kind of join tree of a query  $Q_{1ay}$  equivalent to  $Q$ . That is, the variables of every atom of  $Q$  are contained in an atom of  $Q_{1ay}$  and vice versa. They showed how to find such a tree when there is no disruptive trio. They also showed how, given a database  $D$  over  $Q$ , to construct a database  $D_{1ay}$  over  $Q_{1ay}$  such that  $Q(D) = Q_{1ay}(D_{1ay})$  in  $O(|D| \log |D|)$  time. Then, they showed how to use the special structure of the layered join tree to perform access calls in logarithmic time. In an access call, a fact is selected from each relation of  $D_{1ay}$ , and these facts are joined to form the answer.

We use the same construction and incorporate the annotations as follows. We construct  $Q_{1ay}$  and  $(D_{1ay}, \tau_{1ay})$  from  $Q$  and  $(D, \tau)$ , respectively. For  $Q_{1ay}$  and  $D_{1ay}$ , we use the same construction as that of Carmeli et al. [6], and we apply it by ignoring the annotation. Next, we annotate each fact  $f$  of  $D_{1ay}$  with the initial value  $\tau'(f) = \bar{1}$ , and then apply the following operation.

- 
- 1: **for all** atoms  $\varphi$  of  $Q$  **do**
  - 2:   Select an atom  $\varphi_{1ay}$  of  $Q_{1ay}$  such that  $\text{vars}(\varphi) \subseteq \text{vars}(\varphi_{1ay})$
  - 3:   Let  $R$  and  $R_{1ay}$  be the relation symbols of  $\varphi$  and  $\varphi_{1ay}$ , respectively
  - 4:   **for all** facts  $f_{1ay}$  of  $R_{1ay}$  **do**
  - 5:     Find a fact  $f$  of  $R$  such that  $f$  and  $f_{1ay}$  agree w.r.t.  $\varphi$  and  $\varphi_{1ay}$  (see Definition 3)
  - 6:      $\tau_{1ay}(f_{1ay}) \leftarrow \tau_{1ay}(f_{1ay}) \cdot \tau(f)$
- 

Note that in line 5, at most one corresponding  $f$  exists for every  $f_{1ay}$  since  $\text{vars}(\varphi) \subseteq \text{vars}(\varphi_{1ay})$ . Moreover, from the construction of Carmeli et al. [6] it follows that such an  $f$  necessarily exists (since they apply the full reduction of the Yannakakis [23] algorithm). Finding each  $f$  can be done in constant time by constructing a hash table where each key is a tuple of values assigned to  $\text{vars}(\varphi)$  by  $f$ ; then, for each  $f_{1ay}$ , we project out other variables and search the hash table. In total, this procedure can be done in loglinear time.

The access algorithm extends naturally from before: a fact is selected from each relation of  $D_{1ay}$ , and those are combined to form an answer. The annotation of the answer is the product of the annotations of the selected facts. Retrieving the answer takes logarithmic time, and then we compute the annotation in logarithmic time using a constant number of semiring multiplications. The returned answer is annotated correctly: the annotation is indeed the product of the annotations of the facts of  $D$  that form the answer, and those were multiplied to form the annotation in  $D_{1ay}$ . ◀

From the positive side of Theorem 5, we conclude efficient direct access for ACQs  $Q$ , as long as we can efficiently formulate the aggregate function as an annotation over some logarithmic-time commutative semiring. This is stated in the following corollary of Theorem 5.

► **Corollary 6.** *Consider an ACQ  $Q(\vec{x}, \alpha(\vec{w})) :- \varphi_1(\vec{x}, \vec{y}), \dots, \varphi_\ell(\vec{x}, \vec{y})$  where  $\alpha$  is one of Min, Max, Count, Sum, and Avg.*

1. *If the CQ  $Q'(\vec{x}) :- \varphi_1(\vec{x}, \vec{y}), \dots, \varphi_\ell(\vec{x}, \vec{y})$  is free-connex with no disruptive trio, then direct access for  $Q$  is in  $\langle \text{loglinear}, \text{log} \rangle$ .*
2. *Otherwise, if  $Q$  is also self-join-free, then direct access for  $Q$  is not in  $\langle \text{loglinear}, \text{log} \rangle$ , assuming the HYPERCLIQUE hypothesis (in case  $Q$  is cyclic) and the SparseBMM hypothesis (in case  $Q$  is acyclic).*

**Proof.** For the positive side, we simply apply Theorem 5 with the corresponding semiring. In the case where  $\alpha$  is Avg, we compute Sum and Count separately and divide the results. The negative side carries over from Theorem 2 since a direct access solution for  $Q$  in  $\langle \text{loglinear}, \text{log} \rangle$  is also a direct access solution for  $Q'$  in  $\langle \text{loglinear}, \text{log} \rangle$  if we ignore the aggregated values. ◀

► **Remark 7.** Corollary 6 can be easily extended to support multiple aggregate functions  $\alpha_1(\vec{w}_1), \dots, \alpha_k(\vec{w}_k)$ . For that, we can simply solve the problem for each  $\alpha_i(\vec{w}_i)$  separately, and extract the aggregate values of an answer from the  $k$  data structures that we construct in the preprocessing phase. (Moreover, a practical implementation can handle all aggregate values in the same structure.) ┘

## 4.2 Hardness of Count Distinct

Can we generalize Corollary 6 beyond the stated aggregate functions? The most notable missing aggregate function is **CountD** (count distinct). Next, we show that we *cannot* have similar tractability for count distinct, even in the case of a very simple query, under the *small-universe Hitting Set Conjecture (HSC)* [22]. In HSC, we are given two sets  $\mathcal{U}$  and  $\mathcal{V}$  of size  $N$ , each containing sets over the universe  $\{1, 2, \dots, d\}$ , and the goal is to determine whether  $\mathcal{U}$  contains a set that shares an element with (hits) every set in  $\mathcal{V}$ . HSC states that the problem takes  $N^{2-o(1)}$  time for every function  $d = \omega(\log(N))$ . (In fact, it is conjectured that even a randomized algorithm for this problem needs  $N^{2-o(1)}$  time in expectation [22].)

► **Theorem 8.** *Direct access for  $Q(x, \text{CountD}(y)) :- R(x, w), S(y, w)$  is not in  $\langle \text{loglinear}, \text{log} \rangle$ , assuming HSC.*

**Proof.** Let  $\mathcal{U} = \{U_1, U_2, \dots, U_N\}$  and  $\mathcal{V} = \{V_1, V_2, \dots, V_N\}$  be sets of sets of elements of the universe  $\{1, 2, \dots, d\}$  where  $d = N^c$  for some  $0 < c < 1$ . Indeed,  $d = \omega(\log N)$ , as required by the conjecture. We construct the database  $D$  over  $R$  and  $S$  with the fact  $R(i, j)$  for all  $j \in U_i$  and  $S(i, j)$  for all  $j \in V_i$ .

Next, we assume efficient direct access for  $Q(x, \text{CountD}(y)) :- R(x, z), S(y, z)$ , and use that to solve the hitting-set problem. Each query answer is a pair  $(i, c)$  where  $i$  is the index of a set  $U_i$  from  $\mathcal{U}$  and  $c$  is the number of sets  $V_j$  that  $U_i$  hits. By accessing all query answers, we can check all sets in  $\mathcal{U}$ , one by one, and see whether any is hitting all  $N$  sets. The number of facts in  $D$  is  $O(dN)$ , and the number of query answers (and so the number of access calls) is  $N$ . Hence, direct access for  $Q$  in  $\langle \text{loglinear}, \text{log} \rangle$  would imply a solution to the hitting-set problem in better than  $N^{2-o(1)}$  time. Indeed, for any  $d = O(N^c)$ , we get a solution in  $O(N^{1+c} \cdot \log N)$  time, which contradicts HSC for  $c < 1$ . ◀

Importantly, the reduction used in the proof of Theorem 8 does not involve the order, and hence, the theorem holds true even for direct access without any order requirement.

Despite the above example of intractability, it is important to observe that there are cases where Theorem 5 can be used to compute count distinct: when the size of the domain  $\Omega$  of the distinct elements we count is bounded by a logarithm in the input size  $|D|$ . Such an assumption can be realistic when we count, say, distinct categories from a small ontology (e.g., item categories in a sales context), distinct countries from a small collection of countries, and so on. In such cases, we can compute the exact set of distinct elements, and not just their count, by using the set semiring  $(\mathcal{P}(\Omega), \cup, \cap, \emptyset, \Omega)$  since the operations  $\cup$  and  $\cap$  can be performed in logarithmic time for logarithmic domains.

## 5 Incorporating the Annotation and Aggregation in the Order

The results of the previous section apply when the lexicographic order does not include the computed value, or equivalently when the computed value is last in the head of the query. In this section, we explore the ability to include the computed value earlier in the lexicographic order. To this end, we assume that the underlying commutative semiring has an ordered

domain. When the domain is numerical, we will implicitly assume the natural order without mentioning it. In terms of the computational model, we assume that we can compare two given elements of the domain in time logarithmic in the input.

We first discuss the hardness encountered when we desire to incorporate the computed value in the order. It turns out that this hardness is hit already in extremely simple queries. This is a contrast to the case of Section 4 when this value is excluded from the order.

**Hardness of a CQ\***. Consider the simplest possible Cartesian-product query:  $R \times S$  for unary  $R$  and  $S$ . We wish to have the annotation *first* in the order, hence we have the CQ\*

$$Q_{*\times}(\star, x, y) :- R(x), S(y). \quad (1)$$

The next theorem states that under the *3SUM conjecture*, direct access for  $Q_{*\times}$  is impossible over  $\mathbb{K}$ -database with semirings that gave positive results in the previous section. The *3SUM* conjecture [11, 19] states it takes  $N^{2-o(1)}$  time to determine whether a given set of  $N$  elements from  $\{-N^3, \dots, N^3\}$  contains distinct elements  $a, b, c$  such that  $a + b = c$ .

► **Theorem 9.** *Let  $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$  be one of the counting, numerical, max tropical, or min tropical semirings. Direct access for the CQ\*  $Q_{*\times}$  (of Equation (1)) is not in  $\langle \text{loglinear}, \text{log} \rangle$  over  $\mathbb{K}$ -databases, assuming the *3SUM* conjecture.*

The proof of Theorem 9 shows how to solve *3SUM* using an algorithm for  $Q_{*\times}$ . The proof is nontrivial and is more involved in the case of the counting and numerical semirings, where we use results from number theory [8] to define a homomorphism, such that the operation  $\otimes$  of the semiring represents the numerical addition of *3SUM*.

**Hardness of an ACQ.** Theorem 9 states the hardness of direct access for  $R \times S$  by the order of the annotation. For that, we needed to use the power of the annotation, namely, the annotation of an answer  $(a, b)$  is the product of the annotations of  $R(a)$  and  $S(b)$ . This does not necessarily imply that we have a similar hardness when the computed value is within an ACQ, say using *Count*. For example, direct access for  $Q(\text{Count}(), x, y) :- R(x), S(y)$  is clearly in  $\langle \text{loglinear}, \text{log} \rangle$  since the computed value has no impact (as it is always 1).

Nevertheless, we can show that incorporating the computed value in the order introduces hardness for another fixed ACQ  $Q_c(\text{Count}(), x, y)$ . Moreover, this ACQ is tractable if the order was  $x, y, \text{Count}()$ , due to Theorem 5. We prove it using a reduction from  $Q_{*\times}$  with the counting semiring  $(\mathbb{N}, +, \cdot, 0, 1)$ .

► **Theorem 10.** *There exists a free-connex ACQ  $Q_c(\text{Count}(), x, y)$  such that direct access for  $Q$  is not in  $\langle \text{loglinear}, \text{log} \rangle$ , assuming the *3SUM* conjecture.*

In Theorem 9, we stated hardness for the specific CQ\*  $Q_{*\times}(\star, x, y) :- R(x), S(y)$ , while in Theorem 10 we only claimed the existence of the hard ACQ  $Q_c(\text{Count}(), x, y)$  since the latter is more involved (and we construct it in the proof). The reader might wonder whether we could also phrase Theorem 10 over the Cartesian product  $Q(\text{Count}(), x, y) :- R(x), S(y)$  or alike. Clearly, incorporating *Count* in  $R(x) \times S(y)$  would be meaningless since every answer appears exactly once (and has a count of 1). The next theorem shows that the reason goes deeper: even if we add to  $Q(\text{Count}(), x, y) :- R(x), S(y)$  existential variables, the query remains in  $\langle \text{loglinear}, \text{log} \rangle$ . This statement, in contrast to  $Q(\text{Count}(), x, y) :- R(x), S(y)$ , is nontrivial and requires a proof.

► **Proposition 11.** *For  $Q(\text{Count}(), x, y) :- R(x, w), S(y, z)$ , direct access is in  $\langle \text{loglinear}, \text{log} \rangle$ .*

$R$		$R'$	
$x$	$w$	$x'$	$w'$
a	1	a'	1
b	1	b'	1
b	2	c'	1
c	1	c'	2
c	2	d'	1
c	3	d'	2

 $\longrightarrow$ 

$L$			
$c$	$c'$	$X_c$	$X'_{c'}$
1	1	[a]	[a',b']
1	2	[a]	[c',d']
2	1	[b]	[a',b']
3	1	[c]	[a',b']
2	2	[b]	[c',d']
3	2	[c]	[c',d']

$$Q(\text{Count}(), x, x') :- R(x, w), R'(x', w')$$

■ **Figure 2** Example of the construction in the proof of Proposition 11: direct access for the ACQ  $Q(\text{Count}(), x, x') :- R(x, w), R'(x', w')$ .

**Proof.** For ease of notation, we rename  $Q$  as follows:

$$Q(\text{Count}(), x, x') :- R(x, w), R'(x', w')$$

In the remainder of this proof, we fix an input database  $D$  for  $Q$ . Note that the answers are of the form  $(c \cdot c', a, a')$  where  $c$  is the count of the facts  $R(a, \cdot)$  that have  $a$  as the first element, and  $c'$  is the count of the facts  $R'(a', \cdot)$  that have  $a'$  as the first element.

Let us describe the preprocessing step. First, we compute the number of facts  $R(a, \cdot)$  for every possible value of  $a$ . We use the result to create the set of all counts per possible value of  $x$ , and denote this set by  $C$ . For all  $c \in C$ , we also keep a list  $X_c$  with the set of all values  $a$  that appear  $c$  times in the left column of  $R$ . The list  $X_c$  is sorted so that we can easily locate a given  $a$ . We do the same for  $R'$  to obtain  $C'$  and a sorted list  $X'_{c'}$  for every  $c' \in C'$ .

Next, we create a list  $L$ , where for every pair of counts  $(c, c') \in C \times C'$  it holds the tuple  $(c, c', X_c, X'_{c'})$  where  $X_c$  and  $X'_{c'}$  are represented as *pointers* to the corresponding lists. See Figure 2 for an example of the construction of  $L$  from an input database. We perform direct access on  $L$  sorted by  $c \cdot c'$  and weighted by  $|X_c| \cdot |X'_{c'}|$  using prefix sum, similarly to the way established by Carmeli et al. [6] for a single relation, as we briefly describe next.

We first sort  $L$  by the product of the first two elements of each tuple,  $c \cdot c'$ . Notice that  $L$  indeed represents the answers in the order we desire. For example, if the first fact in  $L$  is  $(c, c', X_c, X'_{c'})$ , then the first  $|X_c| \cdot |X'_{c'}|$  answers have the count  $c \cdot c'$  and assign to  $x$  and  $x'$  the values that occur in  $X_c$  and  $X'_{c'}$ , respectively. We then iterate over  $L$ , and for a tuple index  $i$  we compute the sum of  $|X_c| \cdot |X'_{c'}|$  over all tuples in indices  $1, \dots, i-1$  in  $L$ . We denote this sum by  $l_i$ . Note that  $l_{i+1} > l_i$  for all  $i = 1, \dots, |L|$ .

We now describe the access procedure. Suppose that we are requested to fetch result number  $d$ . We perform a binary search on  $L$  to find the tuple in index  $i$  such that  $l_i < d \leq l_{i+1}$ . Assume that this tuple is  $(c, c', X_c, X'_{c'})$ . The count we return is  $c \cdot c'$ . Next, we need to access the  $(d - l_i)$ th element  $(x, x')$  in  $X_c \times X'_{c'}$ , sorted lexicographically by  $x$  and then by  $x'$ . As in multidimensional arrays, we assign to  $x'$  the element with the index  $(d - l_i) \bmod |X'_{c'}|$  of  $X'_{c'}$ , and we assign to  $x$  the element with the index  $\lfloor \frac{d - l_i}{|X'_{c'}|} \rfloor$  of  $X_c$ .



We now analyze the execution time. Processing each of the  $a$  counts and the  $a'$  counts separately requires only  $O(|D|)$  time. The concern is the time it takes to build and sort the list  $L$ , which might be of size  $|C| \cdot |C'|$ . Assume, without loss of generality, that  $|C| \geq |C'|$ . We claim that  $|R^D| = \Omega(|C|^2)$ . If  $R^D$  has the smallest possible number of facts to result in  $|C|$  distinct counts, then the counts are  $1, \dots, |C|$ . The number of facts in this case is  $\sum_{i=1}^{|C|} i = \frac{|C|(|C|+1)}{2}$ . So,  $|L| = |C| \cdot |C'| \leq |C|^2 = O(|R^D|)$ . We conclude that the algorithm runs in  $O(|D| \log |D|)$  preprocessing time and  $O(\log |D|)$  access time. In conclusion, direct access for  $Q$  is in  $\langle \text{loglinear}, \log \rangle$ , as claimed.  $\blacktriangleleft$

## 5.1 Tractability Condition for General Semirings

So far, we have seen examples where it is intractable to incorporate the computed value in the order. *Not* incorporating it is the same as positioning it last in the lexicographic order. In this section, we show that we can be flexible about the position, to some extent, and pull the computed value back to an earlier position. For example, we show that direct access for the following CQ\* is in  $\langle \text{loglinear}, \log \rangle$  over databases annotated with the numerical semiring.

$$Q(w, x, \star, y, z) :- R(w, x), S(x, y, z), T(y, z) \quad (2)$$

Let  $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$  be a commutative semiring in a domain  $\mathbb{K}$  with an underlying order  $\succeq$ . The semiring is said to be  $\otimes$ -monotone if the function  $f_c$  is monotone for every  $c \in \mathbb{K}$ , where  $f_c : \mathbb{K} \rightarrow \mathbb{K}$  is defined by  $f_c(y) = c \otimes y$ . This means that either  $c \otimes a \succeq c \otimes b$  whenever  $a \succeq b$ , or  $c \otimes a \succeq c \otimes b$  whenever  $b \succeq a$ . All specific semirings that we mention in the paper are  $\otimes$ -monotone. Computationally, we assume that we can determine efficiently (in logarithmic time in the input) whether a given  $c$  is such that the function  $f_c(x) = c \otimes x$  is non-decreasing or non-increasing.

► **Theorem 12.** *Let  $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$  be a  $\otimes$ -monotone logarithmic-time commutative semiring, and  $Q(\vec{x}, \star, \vec{z})$  a free-connex CQ\* with no disruptive trio. If every atom of  $Q$  contains either all variables of  $\vec{z}$  or none of them, then direct access for  $Q$  is in  $\langle \text{loglinear}, \log \rangle$ .*

As an example, consider again the CQ\* of Equation (2). The variables that follow the computed value  $\star$  are  $y$  and  $z$ , and indeed, every atom either contains both  $y$  and  $z$  (as the second and third atoms) or contains none of them (as the first atom). Hence, direct access is tractable according to Theorem 12. As another example, recall the intractable CQ\*  $Q_{\star \times}(\star, x, y) :- R(x), S(y)$  from Theorem 9. Note that this is *not* one of the tractable cases of Theorem 12 since there is an atom that contains  $x$  but not  $y$ . In contrast, Theorem 12 does indicate that whenever  $\star$  is not first, as is the case with  $(x, \star, y)$ , the query is tractable.

Similarly to Corollary 6, we conclude the following corollary for ACQs.

► **Corollary 13.** *Let  $Q(\vec{x}, \alpha(\vec{w}), \vec{z}) :- \varphi_1(\vec{x}, \vec{y}), \dots, \varphi_\ell(\vec{x}, \vec{y})$  be a free-connex ACQ with no disruptive trio. Suppose that  $\alpha$  is one of Min, Max, Count, Sum, and Avg. If every atom in  $Q$  contains either all or none of the variables of  $\vec{z}$ , then direct access for  $Q$  is in  $\langle \text{loglinear}, \log \rangle$ .*

In the next section, we study how additional assumptions on the annotated database can lead to additional opportunities to efficiently incorporate the computed value in the ordering.

## 5.2 Locally Annotated Databases

We have seen in Theorem 9 that even the simple CQ\*  $Q_{\star \times}(\star, x, y) :- R(x), S(y)$  is intractable. This hardness does not necessarily apply to ACQs. For illustration, consider the ACQ

$$Q(\text{Sum}(w), x, y) :- R(x, w), S(y). \quad (3)$$

When translating into an annotated database, we obtain the CQ<sup>\*</sup>  $Q_{\star \times}$  over  $\mathbb{Q}$ -databases annotated by the numerical semiring  $(\mathbb{Q}, +, \cdot, 0, 1)$ . Hence, we translate the problem into an intractable one. Nevertheless, direct access for  $Q$  by  $(\star, x, y)$  is, in fact, in  $\langle \text{loglinear}, \log \rangle$ , as we will show in Theorem 17. This discrepancy stems from the fact that the hardness of  $Q_{\star \times}$  (established in the proof of Theorem 9) relies on the annotation of tuples from both  $R$  and  $S$ . Yet, in our translation, all  $S$ -facts are annotated by 1, and only  $R$ -facts have a nontrivial annotation. The resulting  $\mathbb{K}$ -database is such that every fact is annotated by  $\bar{1}$  (the multiplicative identity), with the exception of one relation. We call such a  $\mathbb{K}$ -database *locally annotated* or *R-annotated* (when we need to specify  $R$ ). We now focus on such databases and show how the assumption of local annotations can be used for efficient access.

In the remainder of this section, we restrict the discussion to queries without self-joins<sup>1</sup> and fix a logarithmic-time commutative semiring  $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$ .

**Full CQ\*s.** We first discuss full CQ\*s (i.e., without existential variables). In the next sections, we also discuss the implications on (non-full) ACQs. We begin with some examples that demonstrate the results that follow later in this section.

► **Example 14.** In some cases, incorporating the annotation in an otherwise tractable order may introduce hardness. Let  $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$  be a logarithmic-time commutative semiring, and consider the full CQ<sup>\*</sup>

$$Q(\vec{x}, \star, \vec{z}) :- R(x_1, x_3), S(x_2, x_3)$$

over  $S$ -annotated  $\mathbb{K}$ -databases. Note that  $Q$  has a disruptive trio if  $x_3$  appears after both  $x_1$  and  $x_2$  in the order. Let us consider orders where this is not the case. The lexicographic order  $(\star, x_2, x_3, x_1)$  is not covered by Theorem 12, as  $x_1$  appears in  $R$ , but  $x_2$  does not. However, we will show in Theorem 17 that, when considering  $S$ -annotated  $\mathbb{K}$ -databases, direct access for  $Q$  by this order is in  $\langle \text{loglinear}, \log \rangle$ , while direct access for  $Q$  by  $(\star, x_1, x_3, x_2)$  is *not* in  $\langle \text{loglinear}, \log \rangle$ .  $\lrcorner$

► **Example 15.** It may also happen that we incorporate the annotation non-trivially and the query remains tractable. Consider the full CQ<sup>\*</sup>

$$Q'(\vec{x}, \star, \vec{z}) :- R(x_1, x_3), S(x_2, x_3), T(x_3)$$

over  $T$ -annotated  $\mathbb{K}$ -databases. Note that this case is a slight variation on Example 14. For any lexicographic order  $(\vec{x}, \star, \vec{z})$ , if  $x_3$  appears after  $x_1$  and  $x_2$ , then  $Q'$  has a disruptive trio and, therefore, direct access for  $Q'$  is not in  $\langle \text{loglinear}, \log \rangle$ . As we show in Theorem 17, for  $Q'$  over  $T$ -annotated  $\mathbb{K}$ -databases, that lack of a disruptive trio is a sufficient condition for tractability. That is, if  $x_3$  does not appear after  $x_1$  and  $x_2$ , then for any aggregate function  $\alpha$  it holds that direct access for  $Q'$  (and for  $Q$ ) is in  $\langle \text{loglinear}, \log \rangle$ .  $\lrcorner$

When dealing with  $R$ -annotated databases, we can replace the annotation of the facts of  $R$  with a new extra attribute, added to  $R$ , and then reason about orders that involve the annotation by considering orders that involve the new attribute instead. To do so, we introduce the following variations of a query and order. Let  $Q$  be a full acyclic CQ<sup>\*</sup> without self-joins. For a relation symbol  $R$  of  $Q$ , we define the *R-deannotation* of  $Q(\vec{x}, \star, \vec{z})$  to be the CQ  $Q_R$  obtained as follows, where we denote by  $\phi_S$  the atom of a relation symbol  $S$ .

<sup>1</sup> In fact, for the algorithm, it suffices that the relation with unrestricted annotations will not appear in more than one atom.



- In the head, replace  $\star$  with a new variable  $y$ .
- If, in addition,  $\text{vars}(\phi_R)$  contains only variables from  $\vec{x}$ , then in the head of  $Q_R$ , advance  $y$  to be immediately after the last variable of  $\phi_R$ .
- For each relation  $S$  of  $Q$ , if  $\text{vars}(\phi_R) \subseteq \text{vars}(\phi_S)$  then concatenate  $y$  to the variable sequence of  $\phi_R$ .

► **Example 16.** Consider the CQ $^*$   $Q'$  of Example 15. The  $R$ -deannotation of  $Q'$  is

$$Q_R(\vec{x}_R, y, \vec{z}_R) :- U(x_1, x_3, y), V(x_2, x_3, y), R(x_3, y)$$

where  $\vec{x}_R$  and  $\vec{z}_R$  are adjustments of  $\vec{x}$  and  $\vec{z}$ : if  $x_3$  is in  $\vec{x}$ , the suffix of  $\vec{x}$  that follows  $x_3$  is moved to the beginning of  $\vec{z}$ .  $\lrcorner$

When  $Q$  is full, we can reduce direct access for  $Q$  to direct access for  $Q_R$ . This is stated in the next theorem. The theorem also says that, whenever the annotation domain contains the natural numbers, this reduction is optimal in the sense that, if we got an intractable  $Q_R$ , then direct access for  $Q$  was hard to begin with.

► **Theorem 17.** *Let  $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$  be a logarithmic-time commutative semiring. Let  $Q$  be a full CQ $^*$  without self-joins and  $Q_R$  the  $R$ -deannotation of  $Q$  for a relation symbol  $R$  of  $Q$ .*

1. *If  $Q_R$  is acyclic and has no disruptive trio, then direct access for  $Q$  is in  $\langle \text{loglinear}, \text{log} \rangle$  on  $R$ -annotated  $\mathbb{K}$ -databases.*
2. *Otherwise, if  $\mathbb{N} \subseteq \mathbb{K}$ , then direct access for  $Q$  is not in  $\langle \text{loglinear}, \text{log} \rangle$  on  $R$ -annotated  $\mathbb{K}$ -databases, assuming the **HYPERCLIQUE** hypothesis (in case  $Q_R$  is cyclic) and the **SparseBMM** hypothesis (in case  $Q_R$  is acyclic).*

We obtain the positive side of the result by treating the annotation as an extra variable that depends functionally on the original variables of  $R$ . Our definition of the  $R$ -deannotation is exactly the *FD-reordered extension* [6] of the query with this extra variable, and the tractability of such an extension is known to imply the tractability of the original query. We note that the negative side of Theorem 17 applies to any domain other than  $\mathbb{N}$ , as long as we can generate infinitely many elements according to the underlying order of the semiring.

► **Example 18.** Theorem 17 gives us a useful tool to analyze the previous examples. Recall that in Example 16 we showed the  $R$ -deannotation of  $Q'$  from Example 15. Since  $y$  appears in every atom, it cannot be part of any disruptive trio. In particular, the disruptive trios of  $Q_R$  are exactly the disruptive trios of  $Q$ . Therefore, given an order  $(\vec{x}, \alpha(w), \vec{z})$ , checking whether direct access for  $Q$  is in  $\langle \text{loglinear}, \text{log} \rangle$  boils down to verifying that  $x_1$ ,  $x_2$  and  $x_3$  do not form a disruptive trio.  $\lrcorner$

**General Queries in the Case of Idempotence.** Next, we extend Theorem 17 beyond full CQ $^*$ s. In some cases, it is sufficient to assemble the tools we already established. Consider the following ACQ:

$$Q(\text{Max}(w_2), x_1, x_2, x_3) :- R(x_1, x_3, w_3), S(x_2, x_3), T(x_3, w_1), U(w_1, w_2) \quad (4)$$

We can solve direct access for  $Q$  using direct access for the CQ $^*$

$$Q'(\star, x_1, x_2, x_3) :- R(x_1, x_3, w_3), S(x_2, x_3), T(x_3, w_1), U'(w_1)$$

over  $U'$ -annotated  $\mathbb{Q}$ -databases and the max tropical semiring. We can then use Lemma 4 to eliminate existential variables and reduce direct access for  $Q'$  to direct access for a full

$\text{CQ}^* Q_{\text{full}}$ . As we later prove in Lemma 19, when the input database for  $Q'$  is  $U'$ -annotated over the max tropical semiring, the suitable database for  $Q_{\text{full}}$  is  $T'$ -annotated for a relation symbol  $T'$  of  $Q_{\text{full}}$ . In our case, we obtain

$$Q_{\text{full}}(\star, x_1, x_2, x_3) :- R(x_1, x_3), S(x_2, x_3), T'(x_3)$$

and a  $T'$ -annotated database. From  $Q_{\text{full}}$  we define  $Q_0$  as the  $T'$ -deannotation of  $Q_{\text{full}}$ .

$$Q_0(y, x_1, x_2, x_3) :- R(x_1, x_3, y), S(x_2, x_3, y), T'(x_3, y)$$

Theorem 2 determines that direct access for  $Q_0$  is in  $\langle \text{loglinear}, \text{log} \rangle$ , and so we can use Theorem 17 to deduce that  $Q_{\text{full}}$  is in  $\langle \text{loglinear}, \text{log} \rangle$  on  $T'$ -annotated  $\mathbb{Q}$ -databases. Therefore, we know from Theorem 5 that direct access for  $Q'$  on  $U$ -annotated  $\mathbb{Q}$ -databases is in  $\langle \text{loglinear}, \text{log} \rangle$  and as a consequence so is direct access for  $Q$ .

As we explain next, the argument above is specific to **Max** and not all aggregate functions since we rely on **Max** being *idempotent*. An operation  $\oplus$  is said to be idempotent if for every  $a$  in the domain  $\mathbb{K}$  we have that  $a \oplus a = a$ . We say that a commutative semiring is an  $\oplus$ -*idempotent semiring* if its addition operation,  $\oplus$ , is idempotent.

Let  $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$  be some logarithmic-time commutative semiring. The process of existential variable elimination using Lemma 4 takes a free-connex  $\text{CQ}^* Q$  and a  $\mathbb{K}$ -database  $(D, \tau)$  and translates it to a full acyclic  $\text{CQ}^* Q'$  and a  $\mathbb{K}$ -database  $(D', \tau')$ . When working over an  $\oplus$ -idempotent semiring, if the input database is locally annotated, then the output database is also guaranteed to be locally annotated. The semirings used for **CountD**, **Min**, and **Max** are  $\oplus$ -idempotent and therefore provide such a guarantee, while the semirings used for **Sum** and **Count** do not. In the case of the query of Equation (4), if the aggregate function was **Sum** instead of **Max**, once we eliminate existential variables, the database for  $Q_{\text{full}}$  is no longer guaranteed to be locally-annotated since projecting out the existential variable  $w_3$  may cause  $R$ , in addition to  $T'$ , to be annotated with values other than  $\bar{1}$ .

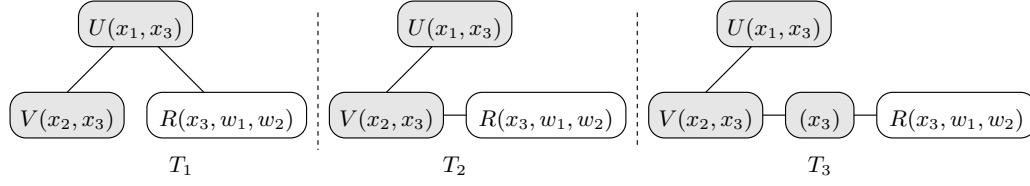
Using Lemma 4 to eliminate existential variables does not guarantee optimal results, even over  $\oplus$ -idempotent semirings. Consider the following simplified version of Equation (4):

$$Q(\text{Max}(w_2), x_1, x_2, x_3) :- U(x_1, x_3), V(x_2, x_3), R(x_3, w_1, w_2) \quad (5)$$

Lemma 4 eliminates existential variables using any ext-free-connex tree and outputs a new full  $\text{CQ}^* Q_{\text{full}}$ . In this process, the vertex of  $R$  is eliminated, and the annotations its relation contained are reflected in a different relation. Then, if Theorem 17 indicates that direct access for  $Q_{\text{full}}$  is in  $\langle \text{loglinear}, \text{log} \rangle$ , we can use  $Q_{\text{full}}$  to provide direct access to  $Q$ . Figure 3 describes three ext-free-connex trees for  $Q$ . The choice between them for usage in Lemma 4 is important:

- If we use the tree  $T_1$ , the annotations would be reflected in the relation of  $U$ . Using Theorem 17 we get that direct access for  $Q_{\text{full}}$  on  $U$ -annotated  $\mathbb{K}$ -databases is not in  $\langle \text{loglinear}, \text{log} \rangle$ . So, this tree cannot be used to obtain direct access for  $Q$  in  $\langle \text{loglinear}, \text{log} \rangle$ .
- If we use the tree  $T_2$ , the annotations would be reflected in the relation of  $V$  and we get that direct access for  $Q_{\text{full}}$  on  $V$ -annotated  $\mathbb{K}$ -databases is not in  $\langle \text{loglinear}, \text{log} \rangle$ . So, this tree cannot be used to obtain direct access for  $Q$  in  $\langle \text{loglinear}, \text{log} \rangle$  either.
- Only by choosing tree  $T_3$  would Lemma 4 admit that direct access to  $Q_{\text{full}}$  is in  $\langle \text{loglinear}, \text{log} \rangle$ . In this case, the new relation that corresponds to  $(x_3)$  would reflect the annotations, and Theorem 17 would indicate that direct access is in  $\langle \text{loglinear}, \text{log} \rangle$ .

With these examples in mind, we establish an effective classification in the case of locally annotated databases over an idempotent semiring. This will be given in Theorem 20. To prove it, we use the next lemma that enables transforming a  $\text{CQ}^*$  with existential variables



■ **Figure 3** Possible ext-free-connex trees of  $Q(x_1, x_2, x_3) :- U(x_1, x_3), V(x_2, x_3), R(x_3, w_1, w_2)$ . The subtree that contains exactly the free variables appears in gray.

into a full CQ<sup>\*</sup>. Note that this lemma is different from Lemma 4 in the sense that the translation preserves intractability in addition to tractability (assuming that the semiring is  $\oplus$ -idempotent).

► **Lemma 19.** *Let  $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$  be a logarithmic-time  $\oplus$ -idempotent commutative semiring. There exists a polynomial time algorithm that takes as input a free-connex CQ<sup>\*</sup>  $Q(\vec{x}, \star, \vec{z})$  without self-joins and a relation symbol  $R$  of  $Q$ , and produces a full acyclic CQ<sup>\*</sup>  $Q'(\vec{x}, \star, \vec{z})$  without self-joins and a relation symbol  $R'$  of  $Q'$ , so that the following are equivalent:*

1. *Direct access for  $Q$  over  $R$ -annotated  $\mathbb{K}$ -databases is in  $\langle \log\text{linear}, \log \rangle$ .*
2. *Direct access for  $Q'$  over  $R'$ -annotated  $\mathbb{K}$ -databases is in  $\langle \log\text{linear}, \log \rangle$ .*

From Lemma 19 we conclude that, to determine the tractability of the CQ<sup>\*</sup>  $Q$ , it suffices to determine the tractability of the CQ<sup>\*</sup>  $Q'$ , which has the property of being full. Using Lemma 19 and Theorem 17, we can now prove our classification of the CQ<sup>\*</sup>s without self-joins over databases locally annotated in the case of an idempotent addition.

► **Theorem 20.** *Let  $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$  be a logarithmic-time  $\oplus$ -idempotent commutative semiring. Let  $R$  be a relation symbol of a free-connex CQ<sup>\*</sup>  $Q$  without self-joins. Let  $Q'$  and  $R'$  be the CQ<sup>\*</sup> and relation symbol obtained from  $Q$  and  $R$  using Lemma 19, and let  $Q_0$  be the  $R'$ -deannotation of  $Q'$ .*

1. *If  $Q_0$  has no disruptive trio, then direct access for  $Q$  over  $R$ -annotated  $\mathbb{K}$ -databases is in  $\langle \log\text{linear}, \log \rangle$ .*
2. *Otherwise, in the case where  $\mathbb{N} \subseteq \mathbb{K}$ , direct access for  $Q$  over  $R$ -annotated  $\mathbb{K}$ -databases is not in  $\langle \log\text{linear}, \log \rangle$ , assuming the *SparseBMM* hypothesis.*

**Proof.** The proof argues about three queries:

1. The input CQ<sup>\*</sup>  $Q(\vec{x}, \star, \vec{z})$ ;
2. A full acyclic CQ<sup>\*</sup>  $Q'(\vec{x}, \star, \vec{z})$ ;
3. A full acyclic CQ  $Q_0$ .

Given  $Q$  and  $R$  as described in Theorem 20, we eliminate existential variables using Lemma 19 and obtain  $Q'$  and a relation symbol  $R'$ . The CQ  $Q_0$  is the  $R'$ -deannotation of  $Q'$ . We can show that  $Q_0$  is acyclic since  $Q'$  is acyclic (and for the exact proof, see the full version of the paper [10]).

From Lemma 19 we conclude that direct access for  $Q$  over  $R$ -annotated  $\mathbb{K}$ -databases is in  $\langle \log\text{linear}, \log \rangle$  if and only if direct access for  $Q'$  over  $R'$ -annotated  $\mathbb{K}$ -databases is in  $\langle \log\text{linear}, \log \rangle$ . So, it remains to prove that direct access for  $Q'$  over  $R'$ -annotated  $\mathbb{K}$ -databases is in  $\langle \log\text{linear}, \log \rangle$  if and only if  $Q_0$  has no disruptive trio.

Suppose first that  $Q_0$  has no disruptive trio. Part 1 of Theorem 17 implies that direct access for  $Q'$  over  $R'$ -annotated  $\mathbb{K}$ -databases is in  $\langle \log\text{linear}, \log \rangle$ , and therefore, so is direct access for  $Q$  over  $R$ -annotated  $\mathbb{K}$ -databases. Conversely, suppose that  $Q_0$  has a disruptive

trio. Suppose also that  $\mathbb{N} \subseteq \mathbb{K}$ , as we assume in Item 2 of Theorem 20. Then Part 2 of Theorem 17 states that direct access for  $Q'$  over  $R'$ -annotated  $\mathbb{K}$ -databases is not in  $\langle \text{loglinear}, \text{log} \rangle$ , assuming SparseBMM. This completes the proof.  $\blacktriangleleft$

Note that it follows from Theorem 20 that, when  $\mathbb{N} \subseteq \mathbb{K}$ , one can determine in polynomial time whether a given CQ\* is in  $\langle \text{loglinear}, \text{log} \rangle$  or not, assuming the SparseBMM hypothesis. Also, as in Corollary 6, we can directly reason about ACQs using Theorem 20. Consider a free-connex ACQ  $Q(\vec{x}, \alpha(\vec{y}), \vec{z}) :- \varphi_1(\vec{x}, \vec{y}), \dots, \varphi_\ell(\vec{x}, \vec{y})$ , where  $\alpha$  is one of CountD over a logarithmic domain, Min, or Max. We can transform the ACQ  $Q$  into a CQ\*  $Q'$ , and then apply Theorem 20 in order to transform  $Q'$  into a CQ  $Q_0$ . If  $Q_0$  has no disruptive trio, then we get direct access for  $Q$  in  $\langle \text{loglinear}, \text{log} \rangle$ . So, at the end of the day, we reduce an ACQ into a CQ\*, which is transformed into a full CQ\* that, in turn, is transformed into a CQ that we solve in  $\langle \text{loglinear}, \text{log} \rangle$ .

**Section summary.** Theorem 12 gives a sufficient condition for tractability of CQ\*s over any logarithmic-time commutative semiring. When inspecting the cases not covered by this sufficient condition, Theorem 9 shows that even simple queries may introduce hardness. Therefore, we narrow our focus to the form of annotation obtained from ACQs (as defined here): locally annotated databases. Theorem 17 shows a dichotomy for full CQ\*s without self-joins. Beyond full CQ\*s, we notice that there are cases, like that of Theorem 10, where the hardness can be attributed to the semiring. We identify the class of  $\oplus$ -idempotent semirings as one that facilitates direct access. In the context of such semirings, Theorem 20 extends the dichotomy of Theorem 17 to support existential variables.

## 6 Concluding Remarks

Direct access provides an opportunity to efficiently evaluate queries even when the number of answers is enormous. Past research studied the feasibility of direct access for CQs, and here we embarked on the exploration of this problem for queries that involve aggregation, either as standard (SQL) aggregate functions or annotation with commutative semirings, that is, CQ\*s and ACQs. We studied the challenges of *construction* and *ordering by* the computed value (aggregation or annotation). We showed that past results carry over to include the computed value, and particularly, that the past classification holds as long as the computed value is not involved in the order. For the second challenge, involving the computed value in the order introduces hardness pretty quickly. We showed a sufficient condition that allows incorporating the computed value in a nontrivial manner. Moreover, we established a full classification of the complexity of CQ\*s without self-joins in the case of databases locally annotated by a semiring with an idempotent addition.

An important direction for future work is the exploration of queries beyond free-connex ones; for that, we need to allow for broader yardsticks of efficiency, as done for direct access for CQs without aggregation [5] and as done for Functional Aggregate Queries (FAQ) for CQs with aggregation and traditional query evaluation [16]. Moreover, we plan to explore the extension of our results to queries with self-joins, and we believe that recent results [5] can be used towards such an extension. It is also left for future work to better understand the limits of computation and establish lower bounds (and dichotomies) for general classes of queries, commutative semirings, and aggregate functions. Another important direction is to explore the ability to efficiently maintain the direct-access structure through updates of the database, as previously studied in the context of non-aggregate queries [3, 21]. Finally, we plan to investigate the practical behavior of our algorithms and understand how well the theoretical acceleration is realized in comparison to existing query engines.

## References

- 1 Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. On acyclic conjunctive queries and constant delay enumeration. In Jacques Duparc and Thomas A. Henzinger, editors, *Computer Science Logic*, pages 208–222, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. doi:10.1007/978-3-540-74915-8\_18.
- 2 Guillaume Bagan, Arnaud Durand, Etienne Grandjean, and Frédéric Olive. Computing the JTH solution of a first-order query. *RAIRO – Theoretical Informatics and Applications (RAIRO: ITA)*, 42:147–164, 2008. URL: <https://hal.archives-ouvertes.fr/hal-00221730>, doi:10.1051/ita:2007046.
- 3 Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering FO+MOD queries under updates on bounded degree databases. *ACM Trans. Database Syst.*, 43(2):7:1–7:32, 2018. doi:10.1145/3232056.
- 4 Johann Brault-Baron. *De la pertinence de l'énumération: Complexité en logiques propositionnelle et du premier ordre*. Theses, Université de Caen, apr 2013. URL: <https://tel.archives-ouvertes.fr/tel-01081392>.
- 5 Karl Bringmann, Nofar Carmeli, and Stefan Mengel. Tight fine-grained bounds for direct access on join queries. In *PODS*, pages 427–436. ACM, 2022. doi:10.1145/3517804.3526234.
- 6 Nofar Carmeli, Nikolaos Tziavelis, Wolfgang Gatterbauer, Benny Kimelfeld, and Mirek Riedewald. Tractable orders for direct access to ranked answers of conjunctive queries. In *PODS*, pages 325–341. ACM, 2021. doi:10.1145/3452021.3458331.
- 7 Nofar Carmeli, Shai Zeevi, Christoph Berkholz, Alessio Conte, Benny Kimelfeld, and Nicole Schweikardt. Answering (unions of) conjunctive queries using random access and random-order enumeration. *ACM Trans. Database Syst.*, 47(3):9:1–9:49, 2022. doi:10.1145/3531055.
- 8 Henri Cohen. *A course in computational algebraic number theory*, volume 138 of *Graduate texts in mathematics*. Springer, 1993. URL: <https://www.worldcat.org/oclc/27810276>.
- 9 Sara Cohen, Werner Nutt, and Yehoshua Sagiv. Deciding equivalences among conjunctive aggregate queries. *J. ACM*, 54(2):5–es, apr 2007. doi:10.1145/1219092.1219093.
- 10 Idan Eldar, Nofar Carmeli, and Benny Kimelfeld. Direct access for answers to conjunctive queries with aggregation. *CoRR*, abs/2303.05327, 2023. doi:10.48550/arXiv.2303.05327.
- 11 Anka Gajentaan and Mark H Overmars. On a class of  $o(n^2)$  problems in computational geometry. *Computational Geometry*, 5(3):165–185, 1995. doi:10.1016/0925-7721(95)00022-2.
- 12 Étienne Grandjean and Louis Jachiet. Which arithmetic operations can be performed in constant time in the ram model with addition?, 2022. doi:10.48550/arXiv.2206.13851.
- 13 Todd J. Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. In *PODS, PODS '07*, pages 31–40, New York, NY, USA, 2007. Association for Computing Machinery. doi:10.1145/1265530.1265535.
- 14 Muhammad Idris, Martin Ugarte, and Stijn Vansummeren. The dynamic yannakakis algorithm: Compact and efficient query processing under updates. In *SIGMOD*, pages 1259–1274, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3035918.3064027.
- 15 Mahmoud Abo Khamis, Ryan R. Curtin, Benjamin Moseley, Hung Q. Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich. Functional aggregate queries with additive inequalities. *ACM Trans. Database Syst.*, 45(4):17:1–17:41, 2020. doi:10.1145/3426865.
- 16 Mahmoud Abo Khamis, Hung Q. Ngo, and Atri Rudra. FAQ: questions asked frequently. In *PODS*, pages 13–28. ACM, 2016. doi:10.1145/2902251.2902280.
- 17 Dan Olteanu and Maximilian Schleich. Factorized databases. *SIGMOD Rec.*, 45(2):5–16, 2016. doi:10.1145/3003665.3003667.
- 18 Dan Olteanu and Jakub Závodný. Size bounds for factorised representations of query results. *ACM Trans. Database Syst.*, 40(1), mar 2015. doi:10.1145/2656335.
- 19 Mihai Patrascu. Towards polynomial lower bounds for dynamic problems. In *Proceedings of the Forty-Second ACM Symposium on Theory of Computing, STOC '10*, pages 603–610, New York, NY, USA, 2010. Association for Computing Machinery. doi:10.1145/1806689.1806772.

## 4:20 Direct Access for Answers to Aggregate Queries

- 20 Christopher Ré and Dan Suciu. The trichotomy of HAVING queries on a probabilistic database. *VLDB J.*, 18(5):1091–1116, 2009. doi:10.1007/s00778-009-0151-4.
- 21 Thomas Schwentick, Nils Vortmeier, and Thomas Zeume. Dynamic complexity under definable changes. *ACM Trans. Database Syst.*, 43(3):12:1–12:38, 2018. doi:10.1145/3241040.
- 22 Virginia Vassilevska Williams. Hardness of easy problems: Basing hardness on popular conjectures such as the strong exponential time hypothesis (invited talk). In *IPEC*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.IPEC.2015.17.
- 23 Mihalis Yannakakis. Algorithms for acyclic database schemes. In *VLDB*, VLDB '81, pages 82–94. VLDB Endowment, 1981.

# Communication Cost of Joins over Federated Data

**Tamara Cucumides**

University of Antwerp, Belgium

**Juan Reutter**

PUC Chile & IMFD Chile, Santiago, Chile

---

## Abstract

---

We study the problem of querying different data sources, which we assume out of our control and that are made available by standard web communication protocols. In this scenario, the time spent communicating data often dominates the time spent processing local queries in each server. Thus, our focus is on algorithms that minimize the communication between the query processing server and the federated servers containing data.

However, any federated query can always be answered with linear communication, simply by requesting all the data to the federated sources. Further, one can show that certain queries do require this amount of communication. But sending all the data is definitely not a relevant algorithm from a practical point of view. This worst-case analysis is, therefore, not useful for our needs. There is a growing body of work in terms of designing strategies that minimize communication in query federation, but these strategies are commonly based in heuristics, and we currently miss a formal analysis providing guidelines for the design of such strategies.

We focus on the communication complexity of federated joins when the problem is parameterized by a measure commonly referred to as the certificate of the instance: a framework that has been used before in the context of set intersection and local query processing. We show how to process any conjunctive query in time given by the certificate of instances. Our algorithm is an adaptation of Minesweeper, one of the algorithms devised for local query processing, into our federating setting. When certificates are of the size of the instance, this amount to sending the entire database, but our strategy provides drastic reductions in the communication needed for queries and instances with small certificates. We also show matching communication lower bounds for cases where the certificate is smaller than the size of active domain of the instances.

**2012 ACM Subject Classification** Theory of computation → Database query processing and optimization (theory); Information systems → Relational database query languages

**Keywords and phrases** databases, database queries, query federation, communication complexity, adaptive algorithms

**Digital Object Identifier** 10.4230/LIPIcs.ICDT.2024.5

**Funding** This work was supported by ANID – Millennium Science Initiative Program – Code ICN17\_002 and FONDECYT Grant No. 1221799.

## 1 Introduction

Federated querying services, in which users can use the Web to access data from different independent sources, are already a part of our current Web architecture. The Semantic Web initiative provides one way of doing this: assuming the data is represented under the RDF standard [12], then these repositories can be queried and merged together using SPARQL, the query language for RDF, by means of the SERVICE operator[4]. Remarkably, the RDF/SPARQL standard also allows for linking non-RDF data, by virtually masking it as intermediate answers of SPARQL queries [15, 22]. To illustrate the uses of web query federation, consider an application in which we recommend city attractions to tourists. Basic information about a city can be automatically obtained from Wikipedia, by querying its public wikidata endpoint at (<https://query.wikidata.org/>). But this information can be



© Tamara Cucumides and Juan Reutter;  
licensed under Creative Commons License CC-BY 4.0  
27th International Conference on Database Theory (ICDT 2024).

Editors: Graham Cormode and Michael Shekelyan; Article No. 5; pp. 5:1–5:19



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



further filtered, for example, by looking at a weather API (so that recommendations agree with current weather), by comments in social networks, or by reviews from platforms such as Yelp or Foursquare. Because we need up-to-date information, queries must be carried out in real time, and the communication must be performed by http or other similar protocol.

We are thus looking to merge information from several web data repositories, which we can only access by means of http requests.

In the context of Web query federation, the main bottleneck in processing queries is not computational power, but web bandwidth usage. Thus, looking for efficiency in the context of query federation means communicating the fewer number of intermediate results between the different data repositories. Furthermore, there is usually an economic incentive for minimizing API requests, as several APIs and endpoints charge for their information.

Consequently, a good deal of research in query federation has been devoted towards algorithms reducing the amount of communication between servers (see e.g. [8, 19] for a good introduction). But so far most approaches rely on heuristics and data profiles, much resembling how traditional DBMS query planning work, and without strict algorithmic properties, nor tools providing guarantees that one approach works better than others.

We believe that current discussion on query federation would benefit from a formal study establishing the limits of what can be done in terms of query processing. Two main questions arising in this context are, first, to understand what are the theoretical limits in terms of communicating tuples (or bits) in query federation. And second, can we design querying strategies that work within these bounds?

Our answers to these questions are based on the framework of adaptive algorithms, as presented in the work of Demaine et al [7]. This framework can be understood as a relaxation of instance-optimal algorithms, wherein one shows that algorithms are optimal for classes of instances that are classified by specific parameters. Adaptive join algorithms have already been studied for traditional (non-federated) query processing in relational databases [16, 10], and we show that these algorithms are a good starting point for the design of algorithms with adaptive communication complexity. We also show that these algorithms are optimal in terms of communication. Up to our best knowledge, our work is the first to introduce the adaptive framework in terms of communication complexity.

## 1.1 Problem Definition and Main Results

Let  $Q(\mathbf{x})$  be a conjunctive query over a schema with relations  $R_1, \dots, R_n$ . A federated instance  $I$  for  $Q$  is a distributed database instance for  $R_1, \dots, R_n$ , in which the interpretation  $R_i^I$  of each relation  $R_i$  resides in a different server. The evaluation  $Q(I)$  of  $Q$  over a federated instance  $I$  is the standard evaluation of  $Q$  over the database instance containing the interpretation of all relations in  $I$ . In web query federation, we wish to materialize all these answers on a separate client, which we abstract as an additional server in our setting. More precisely, we focus on the following problem:

FEDERATED QUERY EVALUATION	
<b>Input:</b>	A conjunctive query $Q$ , a federated instance $I$ for $Q$ .
<b>Task:</b>	Compute $Q(I)$ in a separate server.

As we have mentioned, we assume our servers to be connected only through the Web, and thus the most important bottleneck for Federated Query Evaluation is the amount of communication between each server. Our focus is, then, to solve FEDERATED QUERY EVALUATION using the least amount of communication.



**Efficient algorithms for Federated Query Evaluation.** What do we mean by *efficient* communication? Naturally, the best algorithm should communicate as few information as possible. For a query  $Q$  and a federated instance  $I$ , the bare minimum would be to communicate only the tuples in each relation that participate in  $Q(I)$ , that is, each relation  $R$  in  $Q$  should send only  $|R \times Q(I)|$  tuples. But this is too demanding: if  $Q(I)$  is empty then we would not be allowed to communicate anything at all. Hence, we settle for algorithms that communicate  $|R \times Q(I)|$  tuples per server, plus an extra (small) amount of communication that may depend on  $Q$  and  $I$ .

In traditional join algorithms, the standard is to compute queries in linear time with respect to the database. But asking for *linear communication* is pointless, as the trivial algorithm where we share the entire database already satisfies this bound, even though most of the time this is not a practical option. Our answer builds from the notion of adaptive algorithms devised by Demaine et al. [7], and recently used in the context of (single server) query-processing [16, 10]. Let us first illustrate the idea with an example.

► **Example 1.** Consider query  $Q(x, y, z, w) \leftarrow R(x, y) \wedge S(x, z) \wedge T(x, w)$ . Assuming a federated instance where  $R$ ,  $S$  and  $T$  reside in different servers, computing the answers of  $Q$  using a traditional left-deep plan, or even Yannakakis algorithm [24], would necessarily involve the pairwise intersection of the first components of  $R$ ,  $S$  and  $T$ , in some order. This operation involves the intersection of several sets of elements, which requires a linear amount of communication, as per standard communication complexity results [11]. However, if we intersect all of  $R$ ,  $S$  and  $T$  adaptively, there are several instances for which we can compute the answers of  $Q$  with much less communication: in the extreme case where all elements in (the first components of)  $R$  are smaller than those in  $S$  or  $T$ , we can realize that the answer is empty simply by asking  $R$  for their biggest element, and comparing it to those in  $S$  and  $T$ . Ideally, our algorithms should behave much better in these instances.

The *adaptive* analysis provides a way to measure how complicated are instances for processing queries such as the one in Example 1. More precisely, adaptive algorithms assign to each instance  $I$  a *certificate*, whose size is then used to bound the performance of algorithms. The notion of a certificate for join queries was already defined in [16]: in essence, a certificate is a set of comparisons between elements of the input relations that serves as a warranty for the output of the query. We can use the same ideas in our context of communication, arriving thus at the first goal of our work.

**Goal 1.** Devise an algorithm that solves Federated Query Evaluation, in which each relation  $R_i$  communicates at most  $O(|R_i \times Q(I)| + c)$  tuples, in terms of data complexity<sup>1</sup>, where  $c$  is the size of the smallest certificate for  $I$ , as defined in [16]. In terms of bits of communication, assuming a shared dictionary, **Goal 1** implies communicating  $O((|R_i \times Q(I)| + c) \cdot \log n)$  bits in data complexity, where  $n$  is the size of active domain (the number of different elements) in  $I$ .

We argue that neither left-deep plans nor Yannakakis’s algorithm (for acyclic queries) fulfill the requirements of **Goal 1**, and, up to our best knowledge, neither does any of the known approaches for solving SPARQL Query Federation (see e.g. [19, 5, 6, 14, 13, 18]).

<sup>1</sup> In *data complexity* the size of query  $Q$  is considered to be fixed, and therefore so is the number of attributes in each relation.

Our proposal is to use known adaptive algorithms for query processing, which can be reshaped into well functioning algorithms in the distributed context. More precisely, in Section 4, we show an algorithm for processing join queries that achieves the communication established by **Goal 1**. This algorithm is the adaptation of Minesweeper[16] over the distributed context.

We remark that our focus is on *deterministic* algorithms, in which we compute queries without the need of random inputs and without a probability of error. This is in line with most database algorithms deployed in practice: a vast majority of them are deterministic. Still, understanding the communication implications of randomized algorithms for federated query processing is a very interesting line of research for future work. We do have randomized algorithms with good guarantees for joining two relations [20] and for intersecting sets [3], and these can be deployed as a part of a distributed left-deep plan for processing federated queries. However, this deployment would have the same problem of standard left-deep plan algorithms, where we may end up doing needless intersections because the results of the query do not match somewhere else upstream. Finally, we note that our setting is fundamentally different from the one studied by Beame, Koutris and Suciu [2], where storage is assumed to be shared amongst a big amount of servers. In fact, if we apply results of Beame et al. to our federated instances, these would default to requiring a linear amount of communication between each relation, which, as explained, is not enough for our proposes.

**Proving optimal communication.** The second goal is to show that our algorithms communicate an optimal number of bits. For an instance  $I$  with a domain of  $n$  elements, we assume that the encoding of tuples requires  $O(\log n)$  bits (in data complexity). This is more than what can be achieved by intricate data compressing techniques, but it is much closer to the way words are truly stored on database systems. Thus, if one looks for lower bounds measured in bits, we should focus on lower bounds communicating around  $c \log n$  bits, plus the output of queries. This is our second goal.

**Goal 2.** Prove that any algorithm for Federated Query Evaluation must communicate at least  $\Omega(|R_i \times Q(I)| + c) \cdot \log n$  bits in data complexity, where  $c$  is the certificate size for  $I$ . As it is standard in the literature, we plan to show this by establishing lower bounds on the communication complexity of Federated Query Evaluation. This framework is formally described in Section 2.

We fulfill **Goal 2** for boolean, cyclic queries: given such a query  $Q$ , we can show that it must communicate  $\Omega(c \log n)$  bits, in data complexity, and provide an alternative  $\Omega(\log \binom{n}{c/2})$  bound for boolean acyclic queries. Our construction, however, requires that the size of the certificate is less than the number of elements in the instance, that is, less than the size of the active domain. It is pointless to show a general lower bound for arbitrary queries and larger certificates, as certain queries cannot have bigger certificates. However, we can show a lower bound that applies to infinitely many queries: no matter how large is the certificate and the number of elements in an instance, there is always a boolean query  $Q$  that requires  $\Omega(c \log n)$  bits to be processed. Naturally, these bounds can be directly transferred to non-boolean queries.

**Assumptions in our model.** We assume that instances are made of natural numbers, but our techniques can be extended if one works instead with elements from any other ordered, enumerable domain, provided we have a suitable dictionary for these values. We represent federated instances assuming that every relation resides in a different server, but our scenario

is general enough to model federated queries involving RDF graphs (which are commonly modeled as a single tertiary or quaternary relation), and for cases in which the queries issued to each different server are more complex than just querying a relation, such as with the SPARQL SERVICE operator. In any of these cases, we just assume that the relations in our join query represent a view with the results of a more complex query issued at the particular server. Further, we also assume our queries are connected. When  $Q$  has more than one connected component, then any strategy must involve processing all these components independently, and then combining them using a cross product.

## 2 Preliminaries

**Database basics and notation.** We use  $[n]$  as a shorthand for  $\{1, \dots, n\}$ . Conjunctive queries (CQ) are constructs of the form  $Q(\mathbf{x}) \leftarrow R_1(\mathbf{y}_1) \wedge \dots \wedge R_n(\mathbf{y}_n)$ , where each  $R_i$  is a (not necessarily distinct) relation name, each  $\mathbf{y}_i$  is a tuple of (not necessarily distinct) variables and/or constants, and  $\mathbf{x}$  is a tuple of variables also mentioned in  $\mathbf{y}_1, \dots, \mathbf{y}_n$ . If all such variables are mentioned in  $\mathbf{x}$ , then  $Q$  is *full*. Further, a query  $Q$  is acyclic if it has a join tree, see [24]. We use  $\text{atoms}(Q)$  to refer to the sets of atoms in  $Q$ . The evaluation of a CQ  $Q$  over an instance  $I$  is the set of tuples  $\sigma(\mathbf{x})$ , for each assignment  $\sigma$  from the variables of  $Q$  to elements in  $I$  (and that is the identity on constants), and such that for each atom  $R(\mathbf{y})$  in  $Q$ , the tuple  $\sigma(\mathbf{y})$  is in  $R^I$ . For a full query  $Q$  without constants, we make use of a function  $s_Q(\cdot, \cdot)$  that receives an atom  $R(y_1, \dots, y_k)$  in  $Q$  and an integer  $i \leq k$ , and maps to the corresponding position in  $\mathbf{x} = x_1, \dots, x_n$  such that  $s_Q(R(y_1, \dots, y_k), i) = j$  if  $y_i = x_j$ . Often, both  $Q$  and the atom  $R(y_1, \dots, y_k)$  will be understood from context, so we just denote  $s_Q$  as a unary function  $s$ , as in  $s(i) = j$ .

**Communication Complexity.** Here we just outline the concepts that are necessary for stating our results. We refer to e.g. [11] for a comprehensive treatment on this subject. The (two-way) communication complexity of a function  $f : X \times Y \rightarrow Z$  is defined in terms of *protocols*. Formally, a protocol  $\mathcal{P}$  over  $X \times Y$  with range  $Z$  is a binary tree where each internal node  $v$  is labeled either with a function  $a_v : X \rightarrow \{0, 1\}$  or  $b_v : Y \rightarrow \{0, 1\}$  that indicates how to walk through the tree (right or left) depending on the input, and each leaf is labeled with  $z \in Z$ . The cost of  $\mathcal{P}$  on input  $(x, y)$  is the length of the path taken on input  $(x, y)$  and the cost of  $\mathcal{P}$  is the height of the tree. The communication complexity of  $f$  is the minimum cost of  $\mathcal{P}$ , over all protocols that compute  $f$ .

The communication complexity captures the minimum amount of bits that need to be communicated for a protocol to compute  $f$ . As we have mentioned, our focus is on deterministic protocols. For these, the most widely used technique to prove communication lower bounds are fooling sets, defined next.

► **Definition 2 (Fooling set).** Let  $f : X \times Y \rightarrow \{0, 1\}$ . A set  $S \subset X \times Y$  is called a *fooling set* for  $f$  if there exists  $z \in \{0, 1\}$  such that for every  $(x, y) \in S$ ,  $f(x, y) = z$  and for every distinct pair  $(x_1, y_1)$  and  $(x_2, y_2) \in S$  either  $f(x_1, y_2) \neq z$  or  $f(x_2, y_1) \neq z$

The following result connects fooling sets with communication lower bounds.

► **Lemma 3 ([11]).** If  $f$  has a fooling set of size  $t$ , then the communication complexity of  $f$  is at least  $\log_2 t$

### 3 Communication Complexity of set intersection

To illustrate our framework we start with the simplest join query: the intersection of two unary relations  $R$  and  $S$ . Recall that in our federated setting, both  $R$  and  $S$  reside in different servers, and the answers must be transmitted to a separate master server.

#### 3.1 Adaptive algorithm

Instead of demanding for the full contents of  $R$  and  $S$ , the master can coordinate a sort-merge intersection, by iteratively probing  $R$  for their next element (from the lowest element to the biggest),  $S$  for their next element bigger than that of  $R$ , and so on until all common elements have been found.

As it turns out, this algorithm is optimal in the following, adaptive, way. Consider, for every instance  $I$  over  $R$  and  $S$ , the number of *changes* that we do between  $R$  and  $S$  when both relations  $R$  and  $S$  are sorted onto a single merged list, counting common elements as if they always induce a change.

For example, take  $R = \{2, 3, 5\}$  and  $S = \{1, 3, 5, 7, 9\}$ . When we sort all elements in  $R$  and  $S$ , we start from element 1 (belonging to  $S$ ), then we *change* to 2 (belongs to  $R$ ), then 3, which belongs to both  $R$  and  $S$  and thus we also count, then to 5, which is also counted, then *change* to 7 (only in  $S$ ) and advance without changes to 9, since this element is also only in  $S$ . There are thus four *changes*.

These changes corresponds, informally, to what Demaine et al. coined as the *certificate* for the intersection of  $R$  and  $S$ , and the number of changes corresponds to the *size of the smallest certificate*. The intent of Demaine et al. was to study algorithms that would run in linear time with respect to this size. In our case, we can do the same in terms of communication: when intersecting  $R$  and  $S$  using our coordinated sort-merge intersection, we only communicate  $\theta(c \log n)$  bits, where  $c$  is the size of the smallest certificate.

Note that the size of the smallest certificate for  $R$  and  $S$  ranges from 1 (say, when all elements in  $R$  are smaller than those in  $S$ ) to  $2 \min(|R|, |S|)$  (for example, when  $R$  contains all even numbers up to a given integer  $N$ , and  $S$  contains all odd numbers up to  $N$ ). Thus, asymptotically, this algorithm communicates the same information than just sending  $R$  or  $S$  (up to encoding of elements). But, the smaller this number is for these relations, the smaller the communication issued by our algorithm. This is in concordance with a simple intuitive analysis of this algorithm: in practice it just sounds much better than sending the complete relations.

#### 3.2 Lower bounds

We can also use communication complexity to show that our algorithm is optimal when the certificate size is relatively smaller than the number of elements in  $R$  and  $S$ .

► **Proposition 4.** *The communication complexity of Federated Query Evaluation, on input  $Q(x) \leftarrow R(x) \wedge S(x)$  and an instance with  $n$  elements and certificate size up to  $c$ , is  $\Omega(\log \binom{n}{c/2})$ .*

When  $c$  is smaller than  $n$  (say, bounded by  $n^\epsilon$ , for a fixed  $\epsilon < 1$ ) then  $\log \binom{n}{c/2}$  is  $\Omega(c \log n)$ , which is what we are looking for, as it shows that our algorithm is optimal in terms of communication. While this result does not give the best bounds when certificates are comparable to  $n$ , one important advantage of our sort-merge intersection is that it can be carried out using standard database technology, whereas more nuanced algorithms may involve requests that cannot be processed over a web-based database endpoint.

**Proof of Proposition 4.** We show that the problem of checking whether the boolean version of  $Q$  is empty already requires said communication. Naturally, this is also a lower bound for the non-boolean query  $Q$ , since one can always use the answers of  $Q$  to answer the boolean version. The proof is by building a fooling set: a collection of instances  $I_i = (R^{I_i}, S^{I_i})$ , in such a way that  $Q$  is empty over any such pair, but nonempty over  $(R^{I_i}, S^{I_j})$  for  $j \neq i$ .

Define then, for each set  $A \subseteq [n]$  of size  $|A| = c/2$ , the instance  $I_A$  given by  $R^{I_A} = [n] \setminus A$  and  $S^{I_A} = A$ .

The number of different pairs  $(R^{I_A}, S^{I_A})$  is  $\binom{n}{c/2}$ . Further, we verify that  $R^{I_A} \cap S^{I_A} = \emptyset$ , but  $R^{I_A} \cap S^{I_B} \neq \emptyset$  for  $A \neq B$ , so this collection is indeed a fooling set of size  $\binom{n}{c/2}$ .

It remains to show that all such instances have certificate at most  $c$ . By construction, pairs  $(R^{I_A}, S^{I_A})$  may have up to  $2|A| = c$  changes. Further,  $(R^{I_A}, S^{I_B})$  for  $A \neq B$  require 1 change per element in  $A \cap B$ , and every other element in  $B$  induces at most 2 changes, for a grand total which is always bounded by  $c$ . ◀

The communication bound holds (in data complexity) when intersecting more than two relations. Indeed, for a query  $Q(x) \leftarrow T_1(x) \wedge \dots \wedge T_\ell(x)$ , assume one server contains  $T_1$ , and the other contains all remaining relations. We can then reuse the proof above, setting  $T_1^{I_A}$  as  $R^{I_A}$  and each of  $T_2^{I_A}, \dots, T_\ell^{I_A}$  as  $S^{I_A}$ , to obtain a similar bound<sup>2</sup>. Moreover, this lower bound transfers to the federated multiparty case where each  $T_i$  resides in a different server. If there was an algorithm using less communication to evaluate  $Q$  in the multiparty setting, then we can mimic this algorithm on the two-party setting: every time a relation  $T_i$ ,  $i > 2$  communicates with the master server, in the two party setting this communication happens instead between the master and the server containing relations  $T_2, \dots, T_\ell$ .

## 4 Communication Complexity of natural joins

In this section, we present algorithms for solving Federated Query Evaluation with low communication. We will begin by recalling the extension of certificates to relational queries introduced in [16], and we will follow with the algorithm and its analysis.

### 4.1 Certificates

As in Section 3, the idea is that certificates play the role of *minimal witnesses* for the evaluation of a query over a relational instance.

Let  $R$  be a relation of arity  $k$ . An *index tuple* for  $R$  is a tuple  $(a_1, \dots, a_\ell)$  of  $\ell \leq k$  elements, where each  $a_i$  is either  $-1$  or a natural number.

We use index tuples to produce atoms of the form  $R[a_1, \dots, a_\ell]$ , for  $(a_1, \dots, a_\ell)$  an index tuple. These atoms represent, in each instance  $I$ , an element from a specific tuple in the instantiation  $R^I$  of  $R$  over  $I$ . Specifically, the atom above refers to the tuple in which the first position contains the  $a_1$ -th smallest element amongst all elements in the first position of tuples in  $R^I$ , the  $a_2$ -th smallest element in the second position, and so on.

This is formalized as follows. The interpretation  $R^I[a]$  of  $R[a]$  over an instance  $I$ , for  $a \geq 0$ , is the  $a$ -th smallest value in the first position of tuples in  $R^I$ , i.e., the  $a$ -th smallest value retrieved by the evaluation in  $I$  of query:

$$Q(x_1) \leftarrow R(x_1, \dots, x_k).$$

Further,  $R^I[-1]$  represents the largest element in  $R^I$  retrieved by such query.

<sup>2</sup> Technically speaking, the certificate for these instances is slightly bigger. We give more details in the following section.

## 5:8 Communication Cost of Joins over Federated Data

Next,  $R^I[a_1, \dots, a_{i-1}, a_i]$ , for  $a_i \geq 0$ , represents the  $a_i$ -th smallest element amongst all tuples in  $R^I$  whose first  $i - 1$  elements correspond to  $R[a_1], \dots, R[a_1, \dots, a_{i-1}]$ . That is,  $R^I[a_1, \dots, a_i]$  is the  $a_i$ -th smallest element retrieved by the evaluation over  $I$  of query:

$$Q(x_i) \leftarrow R(R^I[a_1], R^I[a_1, a_2] \dots, R^I[a_1, \dots, a_{i-1}], x_i, \dots, x_k).$$

Likewise,  $R^I[a_1, \dots, a_{i-1}, -1]$  is the largest element retrieved by such query.

We use the notation  $tup^I(R[a_1, \dots, a_\ell])$ , for  $\ell \leq k$ , to refer to the tuple given by  $(R^I[a_1], R^I[a_1, a_2], \dots, R^I[a_1, \dots, a_\ell])$ . In our application we will always have that  $a_i$ , when positive, is at most the number of elements retrieved by the corresponding query. If, in any case, there were less elements, then  $a_i$  will just represent the largest element, as with  $-1$ .

► **Example 5.** Consider an instance  $I$  where

$$R^I = \{(1, 1), (1, 2), (3, 3), (5, 5)\}.$$

Then, interpretation  $R^I[0]$  of  $R[0]$  is element 1, the smallest element in the first position of a tuple in  $R^I$ . Next,  $R^I[0, 1]$  corresponds to element 2: the second smallest element in the second position of a tuple in  $R^I$  that starts with 1. Finally, construct  $R^I[-1, 0]$  is 5: the smallest element in the second position of a tuple in  $R^I$  that starts with  $R^I[-1] = 5$ .

We use  $tup^I(R[0, 1])$  to refer to  $(R^I[0], R^I[0, 1]) = (1, 2)$ .

Index tuples allow us to pinpoint elements in instances, without referring to the actual element, only their ordering within a certain relation. The notion of certificate is then based on comparing two of these atoms.

► **Definition 6** (Argument [16]). *An argument  $\mathcal{A}$  is a set of comparisons of the form:*

$$R[\mathbf{a}] \theta R[\mathbf{b}],$$

with  $a, b$  index tuples  $\theta \in \{<, =, >\}$  and  $R, S$  atoms of the query. An instance  $I$  satisfies an argument if  $R^I[\mathbf{a}] \theta S^I[\mathbf{b}]$  holds in  $I$  for every comparison in  $\mathcal{A}$ .

We are ready to define the notion of certificate. For a full query  $Q$  and an instance  $I$ , a witness for  $Q(I)$  is a set consisting of one atom  $R[a_1, \dots, a_k]$  for each atom  $R(x_1, \dots, x_k)$  in  $Q$ , and such that there exists an output tuple  $\mathbf{t}$  in  $Q(I)$  for which  $tup^I(R[a_1, \dots, a_k]) = (\mathbf{t}_{s(1)}, \dots, \mathbf{t}_{s(k)})$  holds for each atom in  $Q$  (here  $s$  is the function  $s_Q(\cdot, \cdot)$  defined in Section 2).

► **Definition 7** (Certificate [16]). *An argument  $\mathcal{A}$  is a certificate for an instance  $I$  over a query  $Q$  if i)  $I$  satisfies  $\mathcal{A}$ , and ii) for any other instance  $J$  satisfying  $\mathcal{A}$ , the set of witnesses for  $Q(I)$  and  $Q(J)$  coincide. The size of a certificate is the number of comparisons in  $\mathcal{A}$ .*

As we are interested in the size of the smallest certificate, we informally refer to *the size of the certificate* for  $I$  over  $Q$  when we really talk about the size of the smallest certificate for  $I$  over  $Q$ .

► **Example 8.** Consider a query  $Q(x, y, z) \leftarrow R(x, y) \wedge S(x, z) \wedge T(y, z)$ , and an instance  $I$  given by  $R^I = \{(1, 1), (1, 2), (3, 3), (5, 5)\}$ ,  $S^I = \{(1, 1), (2, 2)\}$  and  $T^I = \{(1, 1)\}$ .

The tuple  $(1, 1, 1)$  is the sole output for this query. A possible certificate for  $Q$  consists of the following 8 arguments:

$$\begin{aligned} R[0] &= R[0, 0], & R[0, 0] &= S[0], & S[0] &= S[0, 0], \\ S[0, 0] &= T[0], & T[0] &= T[0, 0], & S[0, -1] &= S[0, 0] \\ S[-1] &< R[1], & T[-1] &< R[0, 1]. \end{aligned}$$

All these arguments, together, imply that the smallest tuple (in lexicographical order) of  $R$ ,  $T$  and  $S$  on any instance satisfying the certificate is always witness for the query. Further, because of the argument  $S[-1] < R[1]$ , the largest element in the first position of  $S$  is smaller than the second element in the first position of  $R$ . This rules out any join where  $x$  is not the element  $R^I[0]$ . Next, argument  $T[-1] < R[0, 1]$  indicates that the largest element in the first position of  $T$  is smaller than the element in the second position of a tuple in  $R$  starting with  $R^I[0]$ . This rules out the possibility of a join in  $y$  apart from  $T[0]$ , conditioned to  $x = R^I[0]$ . Finally, argument  $S[0, -1] = S[0, 0]$  says that there is only one tuple in  $S$  whose value in the first position is  $S[0]$ . This means that the only witness for  $z$ , conditioned to  $x = R^I[0]$  and  $y = T[0]$ , is  $S[0, 0]$ . All of these facts together imply that any instance  $J$  satisfying the certificate cannot have any other answer to  $Q$  apart from  $(R^I[0], R^I[0, 0], T^I[0])$ .

We remark that the certificate need not be linked to the size of the instance, nor to the query output. In general, the size of certificates range from 1 to the size of the instance [16].

**Order of tuples is crucial for certificates.** The other important subtlety of certificates is that the ordering of tuples is crucial for the size of the certificates. To see this, consider query  $Q(x_1, x_2) \leftarrow R(x_1, x_2) \wedge S(x_1, x_2)$ , and an instance  $I$  given by  $R^I = [1] \times [n]$  and  $S^I = [2] \times [n]$ . Then  $R$  and  $S$  do not match on  $x_1$ , and the argument  $R[-1] < S[0]$  is a certificate for  $I$  and  $Q$ . But now, let us assume we have ordered our tuples in the opposite way. Now the query is  $R(x_2, x_1) \wedge S(x_2, x_1)$ , and the interpretations are  $R^I = [n] \times [1]$  and  $S^I = [n] \times [2]$ . The certificate must now include all  $n$  arguments of the form  $R^I[i][-1] < S^I[i][-1]$ .

Our results in this section are consistent with the size of the certificate, given a particular ordering of tuples: communication depends on the certificate size, which again may be higher or lower depending on this ordering.

Khamis et al. have studied notions certificates which do not depend on the ordering of variables [10]. And indeed, one could adapt our algorithms for this notion of certificate. We have decided to work with the original proposal of Ngo et al. [16] because this leads to algorithms that are easily implemented over existing database architectures and systems. In Section 6 we explore an intermediate solution based on certificates for several different orderings of tuples, which, again, can be easily implemented on existing architectures, specially in cases such as RDF, where the arity of relations is low (see e.g. [23, 9, 1]).

## 4.2 Distributed Minesweeper

Minesweeper [16] is an algorithm to process natural join queries that has been shown to run with adaptive time guarantees for classes of queries with acyclicity or bounded treewidth properties. The main idea of Minesweeper consists of repeatedly issuing so-called “probe points”, or tuples of elements, which are then queried to see if they belong to the output of the query or not. Then, either the probe point is a valid output tuple, or else we can exhibit a “gap” around it, indicating that no instance tuple of this relation has elements inside this gap. These gaps are then stored as constraints, so that the next point to probe is such that it does not satisfy any constraints. The algorithm runs until there are no new points to probe.

**The distributed version.** Our distributed version adapts the original algorithm onto the distributed setting, and is designed to process queries regardless of their treewidth. Further, we streamline the communication to avoid a factor that is exponential with respect to the query, that is present in the original Minesweeper. Probe points are now issued in lexicographical ordering, and the algorithm divides them into a probe for each relation.



Further, the master now caches all answers from the relations to avoid issuing the same point two times. We begin by illustrating the algorithm by means of an example, and then follow to present the most important parts. We will finish with the analysis, which is tailored specifically towards the communication, and (because of the way we analyze probe points) requires additional techniques from those in the original Minesweeper [16].

► **Example 9.** Consider again query  $Q(x, y, z) \leftarrow R(x, y) \wedge S(x, z) \wedge T(y, z)$  from Example 8, and the federated instance  $I$  given by the interpretations  $R^I = \{(1, 1), (1, 2), (3, 3), (5, 5)\}$ ,  $S^I = \{(1, 1), (2, 2)\}$  and  $T^I = \{(1, 1)\}$ . The sole output in  $Q(I)$  is  $(1, 1, 1)$ .

In the distributed version of Minesweeper, we start with probe point  $(1, 1, 1)$ . This implies asking all of  $R$ ,  $S$  and  $T$  for pair  $(1, 1)$ . All three servers will return **true**, indicating they contain the pair  $(1, 1)$ , and the algorithm stores this in the cache. Next is  $(1, 1, 2)$ , which implies asking  $R$  for  $(1, 1)$ ,  $S$  for  $(1, 2)$  and  $T$  for  $(1, 2)$ . We will not probe  $R$  this time, because pair  $(1, 1)$  is already in the cache. However, pair  $(1, 2)$  is not in  $S$ , so it returns the constraint  $\langle 1, (2, \infty) \rangle$ , which indicates that there are no tuples of the form  $(1, a)$  in  $S$ , with  $a \in [2, \infty]$ . As explained in Example 8, this information is captured by the argument  $S[0, -1] = S[0, 0]$  in the certificate for  $Q$  and  $I$ . Analogously, relation  $T$  also returns  $\langle 1, (2, \infty) \rangle$ .

The next pair in lexicographical order is  $(1, 1, 3)$ , but because of the constraints in  $S$  and  $T$  we know that no tuple  $(1, 1, b)$ , for  $b > 1$  is in the output of the query. Thus, the next point issued is  $(1, 2, 1)$ .  $R$  has the pair  $(1, 2)$  so we store this in the cache for  $R$ .  $S$  is not queried because  $(1, 1)$  is already cached for  $S$ , and  $T$  is queried for  $(2, 1)$ , returning  $\langle (2, \infty) \rangle$ . As before, this constraint can be matched to an argument in the certificate, in this case to  $T[-1] < R[0, 1]$ . Continuing the search for the next probe point, in lexicographical order, that satisfies all constraints, we arrive at  $(2, 1, 1)$ . Relation  $R$  is probed with pair  $(2, 1)$ , relation  $S$  with  $(2, 1)$  and relation  $T$  is not probed, because  $(1, 1)$  is in the cache of  $T$ . Relation  $R$  does not have  $(2, 2)$ , so it returns the constraints  $\langle (2, 2) \rangle$ . Relation  $S$  returns  $\langle 2, (1, 1) \rangle$ . The next and final point is  $(3, 1, 1)$ . Here  $R$  does return **true**, but  $S$  returns instead the constraint  $\langle (3, \infty) \rangle$ . The last two rounds can be matched to the argument  $S[-1] < R[1]$ , we verify that there are no further matches for variable  $x$ , and finish the search for output tuples.

**The algorithm.** For conciseness, the algorithm is shown for full queries, without self-joins, and that feature more than one relation. This is without loss of generality. If a query is not full, then we first compute the answer for the full query, and then project locally at the master server. We do not consider this a shortcoming of the algorithm, as the lower bounds in Section 5 are always given for boolean queries. Self-joins are treated by packing together all atoms of the same relation  $R$  in a view, probing instead the server for  $R$  for the results of this view. Finally, queries involving only one relation can be dealt with separately by just sending the query to the corresponding server.

We assume that the ordering of variables is preserved between the head and the body of queries: any variable  $x$  that appears before a variable  $y$  in the head  $Q(\mathbf{x})$  of the query, always appears before  $y$  in any of the atoms  $R_i(\mathbf{y}_i)$  in the body. In Section 6 we discuss alternative algorithms that can deal different orderings in atoms.

Algorithms 1 and 2 contain the main parts of the distributed version of Minesweeper. Algorithm 1 requires  $\text{getProbePoint}(\mathbf{t})$  (Algorithm 3 in Appendix A.1), which returns the first tuple that is both greater than  $\mathbf{t}$  (in lexicographical order) and that satisfies the constraints. Once this tuple is generated, the algorithm builds the projection of  $\mathbf{t}$  to the variables of each relation  $R$  of the query, and sends this to  $\text{probe}_R()$ . In turn,  $\text{probe}_R(\mathbf{t})$  checks whether such tuple belongs to  $R$ , and outputs either **true**, if it does, or a constraint if it does not. The algorithm generates the constraints that relate to the most significant position in  $\mathbf{t}$ ; this is



important since we are searching for new probe points in lexicographical ordering<sup>3</sup>. Finally, distributed Minesweeper terminates when there are no more tuples to generate. The final answer is constructed by evaluating  $Q$  over the cached data.

■ **Algorithm 1** Distributed Minesweeper - Master server, query  $Q$ .

---

```

1:  $\mathbf{t} = (0, \dots, 0)$ 
2: while  $\mathbf{t} = \text{getProbePoint}(\mathbf{t}) \neq \text{NULL}$  do
3:   for  $R(x_1, \dots, x_k) \in \text{atoms}(Q)$  do
4:      $\mathbf{t}_R \leftarrow (\mathbf{t}_{s(1)}, \dots, \mathbf{t}_{s(k)})$ 
5:     if  $(\mathbf{t}_R, R)$  is not in the cache then
6:        $\text{return}_R \leftarrow \text{probe}_R(\mathbf{t}_R)$ 
7:       if  $\text{return}_R = \text{true}$  then
8:         store  $(\mathbf{t}_R, R)$  in the cache
9:       else  $(\text{return}_R)$  is a constraint
10:      store  $(\text{return}_R, R)$  as a constraint
11: return evaluation of  $Q$  over the cache.

```

---

■ **Algorithm 2**  $\text{probe}_R(\mathbf{t})$  - server storing  $R$ .

---

```

1:  $k \leftarrow \text{arity}(R)$ 
2: for  $i = 1$  to  $k$  do
3:   if  $R(\mathbf{t}_1, \dots, \mathbf{t}_i, x_{i+1}, \dots, x_k)$  is empty then
4:     if  $R(\mathbf{t}_1, \dots, \mathbf{t}_{i-1}, x_i, \dots, x_k) \wedge x_i > \mathbf{t}_i$  is empty then
5:       return  $\langle \mathbf{t}_1, \dots, \mathbf{t}_{i-1}, (\mathbf{t}_i, \infty) \rangle$ 
6:     else
7:        $e \leftarrow$  smallest element such that  $R(\mathbf{t}_1, \dots, \mathbf{t}_{i-1}, e, x_{i+1}, \dots, x_k) \wedge e > \mathbf{t}_i$  is not
       empty
8:       return  $\langle \mathbf{t}_1, \dots, \mathbf{t}_{i-1}, (\mathbf{t}_i, e - 1) \rangle$ 
9: return true

```

---

**Analysis.** To bound the communication of distributed Minesweeper, we focus on the number of probe points generated by  $\text{getProbePoint}()$  and the communication associated with each one of them. We divide probe points  $\mathbf{t}$  into three types: (1) probe points  $\mathbf{t}$  that are part of the output, (2) probe points  $\mathbf{t}$  for which some  $R$  returns a constraint that can be associated with a comparison in the certificate, and (3) probe points  $\mathbf{t}$  such that all relations either return **true** or a constraint over private attributes (i.e. attributes that appear only in one atom of the query). The following proposition bounds the communication of Distributed Minesweeper. In order to abstract from encoding issues, we give bounds in terms of the number of tuples (or constraints) communicated by the algorithm.

► **Proposition 10.** *The number of tuples communicated by Distributed Minesweeper to each server  $R \in \text{atoms}(Q)$  is in  $O(c + |R \times Q(I)|)$ , in data complexity, where  $c$  is the certificate size for  $I$  over  $Q$ .*

---

<sup>3</sup> In contrast, the original Minesweeper performs a search in the vicinity of  $\mathbf{t}$ . We avoid this search because it may lead to more communication.

Assuming servers use a dictionary for their values, the communication outlined above can be written as  $O((c + |R \times Q(I)|) \cdot \log n)$  if we assume the query processing server also has access to the dictionary, or  $O(c \log n + ||R \times Q(I)||)$  if we do not assume this, where  $||R \times Q(I)||$  is the amount of bits required to encode  $R \times Q(I)$ .<sup>4</sup>

The proof, as we explained before, follows by showing that the number of iterations of the algorithm (i.e. the number of probe points) is at most  $2v(2c + |Q(I)|)$ , where  $c$  is the size of the certificate for  $Q$  and  $I$  and  $v$  is the number of variables in  $Q$ . This number comes up by directly counting probe points of type (1) and (2) and charging probe points of type (3) to the priors. Unlike the original Minesweeper, our probe points are generated in lexicographical order. Although this allows us to simplify the constraints that are returned while roughly keeping the number of probe points, it also changes the way we count such probe points, and especially how we deal with case (3). Finally, when addressing the communication between a relation  $R$  and the master, we only count the first time  $t_R$  is probed into  $R$ , since the result will either be discarded by the constraint returned to this first call, or will be already in our cache.

A natural question at this point is whether the communication bounds exhibited are a property of Minesweeper only, or if they are shared by other worst-case algorithms such as Yannakakis or Leapfrog Trie Join (LFTJ)[21], and what communication guarantees we can achieve (if any) when they are adapted to the federated scenario. Ngo. et al [16] tackled a similar question: whether those worst-case optimal algorithms could run in an instance optimal runtime and the answer was negative for both. Following similar arguments, we can prove that there are instances for which both, Yannakakis and LFTJ need to communicate significantly more tuples, compared to distributed Minesweeper (see Appendix A.2). Let us end this section with a few additional remarks from the point of view of database practice.

**A note on implementation via database queries.** In the context of Web federation, we cannot impose servers to adhere to a protocol of our choosing. In this case, most probably we will only have the ability to issue SQL or SPARQL queries remotely, to the endpoint. One of the advantages of our build is that it can be easily simulated with database queries (lines 3, 4 and 7 in Algorithm 2), and we only pay the cost of issuing (at most)  $k + 1$  queries instead of one call to  $\text{probe}_R$ , and this can be further alleviated with more caching.

**Certificate sizes in real life.** Another important question is what happens for certificates in real life queries. A thorough analysis is out of the scope of this paper, but certificates tend to be much smaller for queries with high selectivity. For example, recall query  $R(x, y) \wedge S(x, z) \wedge T(x, w)$  from the introduction. This query abstracts high-selectivity SPARQL queries that are typical in benchmarks (see e.g. the LUBM benchmark <http://swat.cse.lehigh.edu/projects/lubm/>), such as *retrieve name and address of all professors working in a university*. Here relation  $R$  abstracts the professor-works-in relation, and the other relations abstract personal information from every type of person in the database, and are therefore much bigger. The certificate for this query is of size comparable to the number of elements in the first position of  $R$  (the number of professors). This adequately captures the fact that  $R$  is the relation inducing high-selectivity, and we remark our algorithm achieves this without the need of any heuristics.

---

<sup>4</sup> If the dictionary is not known to the master server, then one needs to retrieve final tuples from the server containing  $R$  with an extra  $|R \times Q(I)|$  requests, which require  $(|R \times Q(I)| \log n + ||R \times Q(I)||)$  bits of communication.

## 5 Lower Bounds

In this section we show lower bounds for the communication complexity of the Federated Query Evaluation problem, for arbitrary conjunctive queries. Recall that our goal is to show a  $\Omega(c \log n)$  lower bound, where  $n$  is the number of elements in the instance, and  $c$  is the certificate size. We begin with communication bounds that are applicable to *any* conjunctive query. For acyclic queries, we cannot do much more than Proposition 4, but cyclic queries allow for a tighter bound that does satisfy our goal.

However, this first result requires that the certificate is at most the size of the number of elements in the instance. It is not possible to push forward a general result applicable to any query, because certain queries cannot have bigger certificates. But for queries of a specific form, we can go much further: we finish this section showing that, no matter the instance and the certificate sizes, there are always queries for which Federated Query Evaluation needs to communicate around  $c \log n$  bits.

For obvious reasons, queries in this section are assumed to contain more than one relation, as otherwise we can always process them locally.

► **Proposition 11.** *The communication complexity of Federated Query evaluation, on input a query  $Q$  (using more than one relation) and an instance  $I$  with at most  $n$  elements and certificate size  $c$  at most  $\frac{n}{2}$  (i.e.  $c \leq \frac{n}{2}$ ) is, in data complexity:*

- $\Omega(\log \binom{n}{c/2})$  bits, if  $Q$  is acyclic, and
- $\Omega(c \log n)$  bits if  $Q$  is cyclic.

**Proof.** For both cases, the proof follows by constructing a fooling set of adequate size. We show the proof for a simpler class of cyclic queries. We leave the general case, as well as the acyclic case, for the full version.

Our class of queries use only binary relations, and we assume that in the cycle of the graph of  $Q$  there is a variable  $x$  that participates in exactly two different atoms, one using relation  $R$  and the other using relation  $S$ . Without loss of generality, we also assume this variable is in the rightmost position in both  $R$  and  $S$ . If this is not the case, one can always reorder all attributes in relations before processing the query in the master server.

As in the proof of Proposition 3, our lower bounds are for two-party communication ( $R$  and  $S$ ), assuming they already know the rest of the database, and this transfers to our multiparty framework without any added communication, in data complexity.

Fix an integer  $k > 0$  for a tuple  $b = b_1, \dots, b_k$ , consider relations  $R_{b_1, \dots, b_k}$  and  $S_{b_1, \dots, b_k}$ , defined as follows.

- $R_{b_1, \dots, b_k}$  is the union of sets of tuples  $\{(i, a) \mid a \neq b_i\}$ , for  $1 \leq i \leq k$ , and
- $S_{b_1, \dots, b_k}$  contains all tuples  $(i, b_i)$ , for  $1 \leq i \leq k$ .

Our fooling set is constructed by instances  $I_{b_1, \dots, b_k}$  in which the interpretation of  $R$  and  $S$  correspond to one of the  $n^k$  pairs of instances  $(R_{b_1, \dots, b_k}, S_{b_1, \dots, b_k})$ , and the interpretation of all other relations in the query contains all  $k$  pairs  $(1, 1), \dots, (k, k)$

The following claim establishes that the set  $\{I_{b_1, \dots, b_k}, b_j \in [n]\}$  is indeed a fooling set for the federated evaluation problem, on input  $Q$ . In turns, this entails that the (deterministic) communication complexity of this problem is at least  $\log n^k = k \log n$ .

▷ **Claim 12.** The evaluation of  $Q$  over any instance  $I_{b_1, \dots, b_k}$  is empty. The evaluation of  $Q$  over any instance whose interpretation are relations  $R_{b_1, \dots, b_k}$  and  $S_{b'_1, \dots, b'_k}$ , where at least one  $b_i$  is different to  $b'_i$ , is not empty.

To finish the proof we need to count the size of the certificates we may form out of our instances, and ensure they satisfy the necessary bounds.

Let us first analyze the certificate for some instance  $I_{b_1, \dots, b_k}$ . For a given  $i \leq k$ , we need arguments  $R[i, b_i - 1] < S[i, 0]$  and  $S[i, -1] < R[i, b_i]$ , plus a series of extra arguments that depend on the query: if the rest of the query uses (binary) relations  $T_1, \dots, T_\ell$ , then we need arguments  $T_j[i] = T_j[i, 0]$ ,  $T_j[i, 0] = T_j[i, -1]$ , for each  $1 \leq j \leq \ell$ , and then  $T_j[i] = T_{j+1}[i]$  for  $1 \leq j \leq \ell - 1$ . Finally, we also need arguments  $R[i] = T_1[i]$  and  $S[i] = T_1[i]$ . The last two arguments ensure that relations contains no more values in its first components:  $R[-1] = R[k]$ ,  $S[-1] = S[k]$ ,  $T_1[-1] = T_1[k]$ . This gives us  $4 + 3\ell$  arguments for each  $1 \leq i \leq k$ , plus three additional ones, for a grand total that is less than  $4k(\ell + 2)$ .

We also need to review the instances obtained by combining two instances from the fooling set. Let us thus build the certificate for the instance grouping  $R_{b_1, \dots, b_k}$ ,  $S_{b'_1, \dots, b'_k}$  and the remaining instances as in any  $I_{b_1, \dots, b_k}$ . Any  $i \leq k$  for which  $b_i = b'_i$  functions exactly like the case above. If  $b_i \neq b'_i$ , then the tuple  $tup(S[i, 0])$  matches with the corresponding tuple in  $R$ . Let us assume that  $b_i > b'_i$ . Then we cover this using arguments  $S[i, 0] = R[i, b'_i]$  and  $S[i, -1] < R[i, b'_i + 1]$ . If  $b_i < b'_i$ , this means that we have a gap in  $R$  before reaching  $b'_i$ , so we use instead arguments  $S[i, 0] = R[i, b'_i - 1]$  and  $S[i, -1] < R[i, b'_i]$ . The rest of the certificate is as before, and thus the certificate for this instance is of the same size.

Summing up, the communication complexity is  $k \log n$ , when given input instances of certificate size at most  $4k(\ell + 2)$ . Choosing  $k = c/(4(2 + \ell))$ , we obtain the required complexity bound: the communication complexity on instances of certificate size at most  $c$  is  $c/(4(2 + \ell)) \log n$ , which is  $\Omega(c \log n)$  in data complexity. ◀

**A second lower bound.** Our next result offers a lower bound for arbitrary certificate sizes, albeit not for any query: the larger the certificate size, the larger the relations must be, as the certificate is bounded by the number of tuples in each relation. The proof of this proposition follows from boosting the number of available tuples used for the proof of Proposition 11.

► **Proposition 13.** *For every  $n, c > 1$  there is a query  $Q$  for which the communication complexity of Federated Query Evaluation on input  $Q$  and instances  $I$  with at most  $n$  elements and certificate size at most  $c$  must communicate  $\Omega(c \log n)$  bits.*

## 6 Moving beyond certificates based on ordering of tuples

There is a close relationship between the way certificates are built and the constraints returned by Minesweeper. At the same time, distributed Minesweeper is modular enough so that any other notions of certificates and constraints can be plugged into our algorithm, without any essential modifications. Khamis et al., joining the team that proposed the original Minesweeper algorithm, studied certificates—and constraints—defined as *gap boxes*, or multidimensional cubes over the space given by all variables participating in a query [10]. Gap boxes make up for a very elegant notion of certificate, which tend to be much smaller than the one we defined in this paper, and would therefore lead to much smaller communication in the distributed version. Unfortunately, working with gap boxes takes us a bit too far away from what can be expected from federated servers; we are mostly stuck with probe algorithms that can easily be expressed in common database query languages. For this reason, and inspired by the *Triple Pattern Fragment* SPARQL Federation infrastructure [23], we study a milder improvement based on incorporating different *orders* for probing relations.

**Pattern Fragment Certificates.** Let  $R$  be a relation of arity  $k$ , and  $o$  be a permutation of  $[k]$ . Then  $o(R^I)$  is the permutation of each tuple in  $R^I$  according to  $o$ , that is, the set of all tuples  $(\mathbf{t}_{o(1)}, \dots, \mathbf{t}_{o(k)})$ , for  $(\mathbf{t}_1, \dots, \mathbf{t}_k)$  in  $R^I$ . Further, for an index tuple  $[a_1, \dots, a_k]$ , the construct  $o(R^I)[a_1, \dots, a_k]$  represent the instantiation of  $[a_1, \dots, a_k]$ , but using  $o(R^I)$  instead of  $R^I$ .

A *pattern fragment* (PF)-argument  $\mathcal{T}$  is a set of comparisons  $R^{o_R}[\mathbf{a}] \theta S^{o_S}[\mathbf{b}]$ , where  $R$  and  $S$  are relations of arities  $k_R$  and  $k_S$ ,  $a, b$  are index tuples,  $\theta \in \{<, =, >\}$  and  $o_R$  and  $o_S$  are permutations of  $[k_R]$  and  $[k_S]$ , respectively. An instance  $I$  satisfies a TPF-argument of the form above if  $o_R(R^I)[\mathbf{a}] \theta o_S(S^I)[\mathbf{b}]$  holds in  $I$  for each such comparison in  $\mathcal{T}$ . The definition of certificate transfers verbatim: a PF-argument  $\mathcal{T}$  is a PF-certificate for an instance  $I$  over a query  $Q$  if  $I$  satisfies  $\mathcal{T}$  and for any other instance  $J$  satisfying  $\mathcal{T}$ , the set of witnesses for  $Q(I)$  and  $Q(J)$  coincide.

► **Example 14.** PF-certificates can be much smaller. For example, recall the query  $Q(x_1, x_2) \leftarrow R(x_2, x_1) \wedge S(x_2, x_1)$  and instance  $I$  given by  $R^I = [n] \times [1]$  and  $S^I = [n] \times [2]$ , as defined in Section 4, which had a certificate with  $n$  arguments. The PF-certificate now consists only of the comparison  $R^o[-1] < S^o[0]$ , where  $o$  is the permutation defining  $o(1) = 2, o(2) = 1$ . This argument state that, when  $R$  and  $S$  are inverted, the largest element in the first position of the inversion of  $R$  is smaller than the smallest element in the first position of the inversion of  $S$ .

**PF-Minesweeper.** In order to adapt our algorithm, we assume that each federated server storing a relation  $R$  of arity  $k$  has made available some of the possible  $k!$  permutations for  $R$ . Triple Pattern Fragments, for example, mandates that all 6 permutations must be made available for RDF graphs, which are stored as triples. Then, instead of issuing  $\text{probe}_R$  on line 7 of Algorithm 1, the master issues a version  $\text{probe}_R^o(o(\mathbf{t}))$  for each permutation  $o$  made available by the server of  $R$ . All of these probes are of course answered by the server storing  $R$ , and the response is the expected response for a server storing  $o(R^I)$  receiving  $(o(\mathbf{t}))$ . With a little care, all the constraints returned by such algorithms can be combined together, and  $\text{getProbePoint}()$  can be made to work just as before. The resulting communication is similar to what we had before, except we replicate the whole probing process for each available permutation. Since the number of permutations ultimately depend on the atoms used in the query, we arrive at the same data complexity communication bounds.

► **Proposition 15.** *The number of tuples communicated by PF-Minesweeper to each server  $R \in \text{atoms}(Q)$  is in  $O(c + |R \times Q(I)|)$ , in data complexity.*

**What have we gained?** In terms of combined complexity, we can now reduce the number of calls to depend only on the size of the relations, and not on the total number of variables in the query; this is because we can now deal with self-joins directly instead of packing them into views. Further, as more permutations are made available, certificates can only be smaller, and thus the communication in this respect diminishes. However, more orders require more communication in each step, as each relation must be probed once per each permutation available. We believe that this trade-off is worthy of future study, as it may provide practical guidelines for future implementation of web query federation strategies.

## 7 Conclusions and future work

Query federation systems have already been implemented for a few years now, but our work is the first to provide a formal framework in which one can analyze federation strategies without resorting to heuristics or probabilistic distributions. Naturally, our work would best be deployed in a context where certificates can be small. Yet, highlighting the theoretical importance of certificates in the context of web federation is already an important contribution, and our work can guide the design of federated strategies even in context where certificates sizes are comparable to the size of instances.

As for distributed minesweeper, we expect it to shine the most on contexts where queries involve several joins, such as in graphs. The reason we expect this is partly because this has been the case for worst-case optimal algorithms in general [17], and partly because we expect that a distributed version of Yannakakis (using sort-merge join for pointwise semijoins) would probably perform reasonably in practice, even if we know of pathological cases where the communication is much worse. We are looking forward to a prototype implementation of our algorithms, to help us understand how much do these theoretical results transfer to practice.

Lastly, we would like to incorporate randomized protocols into our framework, in which actions can be dictated by a random string. Most communication protocols can be improved when randomization is allowed, so it may be the case that we can do this for database queries as well. Proving lower bounds in this case would involve orchestrating distributions and certificates of instances, an interesting problem for future research.

---

### References

- 1 Diego Arroyuelo, Aidan Hogan, Gonzalo Navarro, Juan L Reutter, Javiel Rojas-Ledesma, and Adrián Soto. Worst-case optimal graph joins in almost no space. In *Proceedings of the 2021 International Conference on Management of Data*, pages 102–114, 2021. doi:10.1145/3448016.3457256.
- 2 Paul Beame, Paraschos Koutris, and Dan Suciu. Communication steps for parallel query processing. *Journal of the ACM (JACM)*, 64(6):1–58, 2017. doi:10.1145/3125644.
- 3 Joshua Brody, Amit Chakrabarti, Ranganath Kondapally, David P Woodruff, and Grigory Yaroslavtsev. Beyond set disjointness: the communication complexity of finding the intersection. In *Proceedings of the 2014 ACM symposium on Principles of distributed computing*, pages 106–113, 2014. doi:10.1145/2611462.2611501.
- 4 Carlos Buil-Aranda, Marcelo Arenas, Oscar Corcho, and Axel Polleres. Federating queries in sparql 1.1: Syntax, semantics and evaluation. *Journal of Web Semantics*, 18(1):1–17, 2013. doi:10.1016/j.websem.2012.10.001.
- 5 Carlos Buil-Aranda, Axel Polleres, and Jürgen Umbrich. Strategies for executing federated queries in sparql1. 1. In *International Semantic Web Conference*, pages 390–405. Springer, 2014. doi:10.1007/978-3-319-11915-1\_25.
- 6 Diego Calvanese, Benjamin Cogrel, Sarah Komla-Ebri, Roman Kontchakov, Davide Lanti, Martin Rezk, Mariano Rodríguez-Muro, and Guohui Xiao. Ontop: Answering sparql queries over relational databases. *Semantic Web*, 8(3):471–487, 2017. doi:10.3233/SW-160217.
- 7 Erik D Demaine, Alejandro López-Ortiz, and J Ian Munro. Adaptive set intersections, unions, and differences. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 743–752, 2000. URL: <http://dl.acm.org/citation.cfm?id=338219.338634>.
- 8 Aidan Hogan. Web of data. In *The Web of Data*, pages 15–57. Springer, 2020. doi:10.1007/978-3-030-51580-5.



- 9 Aidan Hogan, Cristian Riveros, Carlos Rojas, and Adrián Soto. A worst-case optimal join algorithm for sparql. In *International Semantic Web Conference*, pages 258–275. Springer, 2019. doi:10.1007/978-3-030-30793-6\_15.
- 10 Mahmoud Abo Khamis, Hung Q Ngo, Christopher Ré, and Atri Rudra. Joins via geometric resolutions: Worst case and beyond. *ACM Transactions on Database Systems (TODS)*, 41(4):1–45, 2016. doi:10.1145/2967101.
- 11 Eyal Kushilevitz. Communication complexity. In *Advances in Computers*, volume 44, pages 331–360. Elsevier, 1997. doi:10.1016/S0065-2458(08)60342-3.
- 12 Brian McBride. The resource description framework (rdf) and its vocabulary description language rdfls. In *Handbook on ontologies*, pages 51–65. Springer, 2004.
- 13 Gabriela Montoya, Hala Skaf-Molli, and Katja Hose. The odyssey approach for optimizing federated sparql queries. In *International semantic web conference*, pages 471–489. Springer, 2017. doi:10.1007/978-3-319-68288-4\_28.
- 14 Gabriela Montoya, Hala Skaf-Molli, Pascal Molli, and Maria-Esther Vidal. Federated sparql queries processing with replicated fragments. In *International Semantic Web Conference*, pages 36–51. Springer, 2015. doi:10.1007/978-3-319-25007-6\_3.
- 15 Matthieu Mosser, Fernando Pieressa, Juan Reutter, Adrián Soto, and Domagoj Vrgoč. Querying apis with SPARQL: language and worst-case optimal algorithms. In *European Semantic Web Conference*, pages 639–654. Springer, 2018. doi:10.1007/978-3-319-93417-4\_41.
- 16 Hung Q Ngo, Dung T Nguyen, Christopher Re, and Atri Rudra. Beyond worst-case analysis for joins with minesweeper. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 234–245, 2014. doi:10.1145/2594538.2594547.
- 17 Dung Nguyen, Molham Aref, Martin Bravenboer, George Kollias, Hung Q Ngo, Christopher Ré, and Atri Rudra. Join processing for graph patterns: An old dog with new tricks. In *Proceedings of the GRADES’15*, pages 1–8. Association for Computing Machinery, 2015. doi:10.1145/2764947.2764948.
- 18 Muhammad Saleem, Alexander Potocki, Tommaso Soru, Olaf Hartig, and Axel-Cyrille Ngonga Ngomo. Costfed: Cost-based query optimization for sparql endpoint federation. *Procedia Computer Science*, 137:163–174, 2018. doi:10.1016/j.procs.2018.09.016.
- 19 Andreas Schwarte, Peter Haase, Katja Hose, Ralf Schenkel, and Michael Schmidt. Fedx: Optimization techniques for federated query processing on linked data. In *International semantic web conference*, pages 601–616. Springer, 2011. doi:10.1007/978-3-642-25073-6\_38.
- 20 Dirk Van Gucht, Ryan Williams, David P Woodruff, and Qin Zhang. The communication complexity of distributed set-joins with applications to matrix multiplication. In *Proceedings of the 34th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 199–212, 2015. doi:10.1145/2745754.2745779.
- 21 Todd L Veldhuizen. Leapfrog triejoin: a worst-case optimal join algorithm. *arXiv preprint arXiv:1210.0481*, 2012. doi:10.48550/arXiv.1210.0481.
- 22 Ruben Verborgh, Thomas Steiner, Davy Van Deursen, Sam Coppens, Joaquim Gabarró Vallés, and Rik Van de Walle. Functional descriptions as the bridge between hypermedia apis and the semantic web. In *Proceedings of the third international workshop on restful design*, pages 33–40, 2012. doi:10.1145/2307819.2307828.
- 23 Ruben Verborgh, Miel Vander Sande, Olaf Hartig, Joachim Van Herwegen, Laurens De Vocht, Ben De Meester, Gerald Haesendonck, and Pieter Colpaert. Triple pattern fragments: a low-cost knowledge graph interface for the web. *Journal of Web Semantics*, 37:184–206, 2016. doi:10.1016/j.websem.2016.03.003.
- 24 Mihalis Yannakakis. Algorithms for acyclic database schemes. In *VLDB*, volume 81, pages 82–94, 1981.

## A Appendix

### A.1 Details from Section 4.2

For an atom  $R(y_1, \dots, y_k)$ , and a constraint  $\alpha = \langle a_1, \dots, a_\ell, (l, h) \rangle$ , the pair  $(\alpha, R)$  dominates a tuple  $\mathbf{t}$  if  $a_i = \mathbf{t}_{s(i)}$  for each  $1 \leq i \leq \ell - 1$ , and  $l \leq \mathbf{t}_{s(\ell)} \leq h$ . A dominating constraint-relation pair is *infinite* when  $h = \infty$ . Algorithm 3 now presents how we compute the next probe point.

■ **Algorithm 3** `getProbePoint( $\mathbf{t}$ )`.

---

```

1:  $n \leftarrow \text{arity}(\mathbf{t})$ 
2: if  $\mathbf{t} = -1^n$  then
3:   return  $1^n$ 
4:  $\mathbf{t} \leftarrow (\mathbf{t}_1, \dots, \mathbf{t}_{n-1}, \mathbf{t}_n + 1)$ 
5: while true do
6:   if  $\mathbf{t}$  is not dominated by any stored constraint-relation pair then
7:     return  $\mathbf{t}$ 
8:    $i \leftarrow$  the smallest position of a pair  $(l, h)$  in a constraint dominating  $\mathbf{t}$ 
9:   if  $h = \infty$  then
10:    if  $i = 1$  then
11:      return NULL
12:     $\mathbf{t}_j = 1$  for each  $j \geq i$ 
13:     $\mathbf{t}_{i-1} = \mathbf{t}_{i-1} + 1$ 
14:  else
15:     $e \leftarrow$  The highest element  $h$  in pairs  $(l, h)$  in every constraint  $\langle \mathbf{t}_1, \dots, \mathbf{t}_{i-1}, (l, h) \rangle$ .
16:     $\mathbf{t}_i = e + 1$ 
17:     $\mathbf{t}_j = 1$  for each  $j > i$ 

```

---

### A.2 Counterexample for distributed Yannakakis and LFTJ

The following instance is presented in [16] to show that both Yannakakis and LFTJ cannot achieve instance-optimal runtime: this instance also provides an example where LFTJ communication is non-optimal.

Consider the query

$$Q \leftarrow R_1(a_1, a_2) \wedge R_2(a_2, a_3) \dots \wedge R_m(a_m, a_{m+1}).$$

and the instance  $I_{m,M}$  with  $m \geq 5$  and  $M$  an integer, in which each  $R_i^I$  consists in exactly  $m$  blocks where the  $j$ -th block will be defined as follows

- for  $j = i$  the block is just the tuple  $\{(j-1)M + 1, (j-1)M + 1\}$
- for  $j = i - 1$  (or  $m$  when  $i = 1$ ) the block is empty
- for  $j \in [k] - \{i, i - 1\}$  the  $j$ -th block is

$$[(j-1)M + 2, jM] \times [(j-1)M + 2, jM]$$

The output of this instance is empty and it was shown in [16] that this instance has a certificate of size  $O(mM)$  that consists on the following comparisons:

$$\begin{aligned}
R_1[1, 1] &< R_2[1] \\
R_1[i, 1] &> R_2[1], \text{ for } i > 1 \\
R_2[i, 1] &> R_3[M + 1], \text{ for } i > 1 \\
&\vdots \\
R_{m-1}[i, 1] &> R_m[(m-2)M + 1], \text{ for } i > 1
\end{aligned}$$



As per Proposition 10 distributed minesweeper will need to communicate  $O(mM)$  tuples with each  $R$  to compute the answer. Let's see now how can we solve this query with (a distributed version of the) LFTJ. Take an arbitrary attribute ordering  $A_{i_1}, \dots, A_{i_{n+1}}$  and focus on the first two attributes  $A_{i_1}$  and  $A_{i_2}$ :

In order to be able to compare the communication complexity of both our distributed Minesweeper and a federated version of LFTJ we will endow the latter with a cache and communication-free semi-join reductions and let's consider two cases:

1. The first one is when  $|i_1 - i_2| = 1$ . In this case, the algorithm will compute the reduction of the corresponding relation  $R(A_{i_1}, A_{i_2})$  on  $A_{i_1}$  and  $A_{i_2}$  which we assume involves no communication. But after that it needs to go through (and communicate) all tuples in the reduced relation that is of size  $\Omega(mM^2)$
2. On the other hand, when  $|i_1 - i_2| > 1$  then the first thing is to compute the intersections on both attributes  $A_{i_1}$  and  $A_{i_2}$  and perform a cross-product. Both intersections are of size  $\Omega(mM)$  and even though communicating the tuples to compute the cross-product itself is not an issue, in the next step we will necessarily communicate  $\Omega(mM^2)$

Now let's move into the Yannakakis algorithm: the idea here is to exploit the fact that Yannakakis needs to perform semi-joins. Consider for example the following acyclic query  $Q$

$$Q \leftarrow S(x) \wedge R(x, y) \wedge U(y, w) \wedge T(x, z) \wedge V(z, p).$$

And, for each integer  $m$ , the instance  $I_m$  in which  $S^{I_m} = [m]$ ,  $T^{I_m} = [1]x[m]$ ,  $R^{I_m} = [2]x[n]$ , and  $U^{I_m} = V^{I_m} = [m]x[a]$ . Notice that the certificate of  $I_m$  is always of size 1, comprising the sole argument  $T[-1] < R[0]$ .

While there are several version of the Yannakakis algorithm, all versions we are aware of involve a bottom-up reduction of the database. This implies computing the semijoin between  $R$  and  $U$  and  $T$  and  $V$ . However, computing  $R \bowtie U$  or  $U \bowtie R$  involves the intersection of the second component of  $R$  with the first component of  $U$ , which requires the communication of  $n$  bits, as per standard communication complexity results (see e.g. [11]).


The results above can be summarized into the following observations:

► **Observation 16.** *There is an acyclic query  $Q$  and a family  $(I_m)$ ,  $m > 1$  of instances whose certificate for  $Q$  is of size  $O(mM)$ , but for which the Leapfrog trie join algorithm must necessarily communicate  $O(mM^2)$  bits.*

► **Observation 17.** *There is an acyclic query  $Q$  and a family  $(I_m)$ ,  $m > 1$  of instances whose certificate for  $Q$  is of size 1, but for which the Yannakakis algorithm must necessarily communicate  $m$  bits.*



# Range Entropy Queries and Partitioning

Sanjay Krishnan ✉ 

Department of Computer Science, University of Chicago, IL, USA

Stavros Sintos ✉ 

Department of Computer Science, University of Illinois at Chicago, IL, USA

---

## Abstract

Data partitioning that maximizes or minimizes Shannon entropy is a crucial subroutine in data compression, columnar storage, and cardinality estimation algorithms. These partition algorithms can be accelerated if we have a data structure to find the entropy in different subsets of data when the algorithm needs to decide what block to construct. While it is generally known how to compute the entropy of a discrete distribution efficiently, we want to efficiently derive the entropy among the data items that lie in a specific area. We solve this problem in a typical setting when we deal with real data, where data items are geometric points and each requested area is a query (hyper)rectangle. More specifically, we consider a set  $P$  of  $n$  weighted and colored points in  $\mathbb{R}^d$ . The goal is to construct a low space data structure, such that given a query (hyper)rectangle  $R$ , it computes the entropy based on the colors of the points in  $P \cap R$ , in sublinear time. We show a conditional lower bound for this problem proving that we cannot hope for data structures with near-linear space and near-constant query time. Then, we propose exact data structures for  $d = 1$  and  $d > 1$  with  $o(n^{2d})$  space and  $o(n)$  query time. We also provide a tune parameter  $t$  that the user can choose to bound the asymptotic space and query time of the new data structures. Next, we propose near linear space data structures for returning either an additive or a multiplicative approximation of the entropy. Finally, we show how we can use the new data structures to efficiently partition time series and histograms with respect to entropy.

**2012 ACM Subject Classification** Theory of computation → Data structures design and analysis

**Keywords and phrases** Shannon entropy, range query, data structure, data partitioning

**Digital Object Identifier** 10.4230/LIPIcs.ICDT.2024.6

**Related Version** *Full Version:* <https://arxiv.org/abs/2312.15959> [27]

## 1 Introduction

Discrete entropy is defined as the expected amount of information needed to represent an event drawn from a probability distribution. That is, given a probability distribution  $\mathcal{D}$  over the set  $\mathcal{X}$ , the entropy is defined as  $H(\mathcal{D}) = -\sum_{x \in \mathcal{X}} \mathcal{D}(x) \cdot \log \mathcal{D}(x)$ . The entropy has a few different interpretations in information theory and statistics, such as:

- (Compression) Entropy is a lower-bound on data compressibility for datasets generated from the probability distribution via the Shannon source coding theorem.
- (Probability) Entropy measures a probability distribution’s similarity to a uniform distribution over the set  $\mathcal{X}$  on a scale of  $[0, \log |\mathcal{X}|]$ .

Because of these numerous interpretations, entropy is a highly useful optimization objective. Various algorithms, ranging from columnar compression algorithm to histogram construction and data cleaning, maximize or minimize (conditional) entropy as a subroutine. These algorithms try to find high or low entropy data subsets. Such algorithms can be accelerated if we have a data structure to efficiently calculate the entropy of different subsets of data. However, while it is known how to compute the entropy of an entire distribution efficiently, there is a little work on such “range entropy queries”, where we want to derive efficiently the entropy among the data items that lie in a specific area. To make this problem more concrete, let us consider a few examples.



© Sanjay Krishnan and Stavros Sintos;  
licensed under Creative Commons License CC-BY 4.0  
27th International Conference on Database Theory (ICDT 2024).

Editors: Graham Cormode and Michael Shekelyan; Article No. 6; pp. 6:1–6:21

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

► **Example 1** (Columnar Compression). An Apache Parquet file is a columnar storage format that first horizontally partitions a table into row groups, and then applies columnar compression along each column within the row group. A horizontal partitioning that **minimizes the entropy** within each partition can allow for more effective columnar compression.

► **Example 2** (Histogram Construction). Histogram estimation often uses a uniformity assumption, where the density within a bucket is modeled as roughly uniform. A partitioning that **maximizes the entropy** within each partition can allow for more accurate estimation under uniformity assumptions.

► **Example 3** (Data Cleaning). As part of data exploration, a data analyst explores different subsets of data to find areas with high entropy/uncertainty. Usually, subsets of data or items in a particular area of the data with high entropy contain dirty data so they are good candidates for applying data cleaning methods. For example, Chu et al. [16] used an entropy-based scheduling algorithm to maximize the uncertainty reduction of candidate table patterns. Table patterns are used to identify errors in data.

The first two problems above have a similar structure, where an outer-algorithm leverages a subroutine that identifies data partitions that minimize or maximize entropy. In the third problem we aim to explore areas with high entropy by running arbitrary range entropy queries. We formulate the problem of range entropy query problem in a typical and realistic setting when we deal with real data: we assume that each item is represented as a point in the Euclidean space. More specifically, we consider a set  $P$  of  $n$  weighted and colored points in  $\mathbb{R}^d$ . The goal is to construct a data structure, such that given a query (hyper)rectangle  $R$ , compute the entropy of the points in  $P \cap R$  (denoted by  $H(P \cap R)$ ). The entropy of  $P \cap R$  is defined as the entropy of a discrete distribution  $\mathcal{D}_R$  over the colors in  $P \cap R$ : Let  $U_R$  be the set of all colors of the points in  $P \cap R$ . For each color  $u_j \in U_R$ , we define a value (we can also refer to it as an independent event or outcome)  $\alpha_j$  with probability  $w_j$  equal to the sum of weights of points with color  $u_j$  in  $P \cap R$  divided by the sum of the weights of all points in  $P \cap R$ . Notice that  $\sum_{u_j \in U_R} w_j = 1$ . Unfortunately, we do not have direct access to this distribution; we would need  $\Omega(n)$  time to construct the entire distribution  $\mathcal{D}_R$  in the query phase. Using the geometry of the points along with key properties from information theory we propose data structures to find the entropy of  $\mathcal{D}_R$  without constructing  $\mathcal{D}_R$  explicitly.

► **Definition 4** (Range entropy query problem). *Given a set  $P$  of  $n$  weighted and colored points in  $\mathbb{R}^d$ , the goal is to construct a data structure with low space such that given any query rectangle  $R$ , it returns  $H(P \cap R)$  in sub-linear time  $o(|P|)$ .*

If the number of colors in  $P$  is bounded by a constant then the range entropy query problem can be easily solved. However, in the worse case the number of different colors is  $O(n)$ . Our goal is to construct data structures whose query time is always sublinear with respect to  $n$ .

**Summary of Results.** One of the main challenges with range entropy queries is that entropy is not a *decomposable* quantity. Let  $P_1, P_2$  be two sets of points such that  $P_1 \cup P_2 = P$  and  $P_1 \cap P_2 = \emptyset$ . If we know  $H(P_1), H(P_2)$  there is no straightforward way to compute  $H(P_1 \cup P_2)$ . In this paper, we build low space data structures such that given a rectangle  $R$ , we visit points or subsets of points in  $P \cap R$  in a particular order and carefully update the overall entropy. All our results for the range entropy problem can be seen in Table 1.

- In Section 2 we introduce some useful notation and we revisit a way to update the entropy of the union of two sets with no color in common in  $O(1)$  time.

■ **Table 1** New results (lower bound in the first row and data structures with their complexities in the next rows).  $t \in [0, 1]$  is a tune parameter.  $\tilde{O}(\cdot)$  notation hides a  $\log^{O(1)} n$  factor, where the  $O(1)$  exponent is at most linear on  $d$ .  $Q(n)$  is any function of  $n$  that represents the query time of a data structure storing  $n$  items.

Type	Space	Query Time	Preprocessing
Lower bound	$\tilde{\Omega}\left(\left(\frac{n}{Q(n)}\right)^2\right)$	$\tilde{O}(Q(n))$	–
$d = 1$ , exact	$O(n^{2(1-t)})$	$\tilde{O}(n^t)$	$O(n^{2-t})$
$d > 1$ , exact	$\tilde{O}(n^{(2d-1)t+1})$	$\tilde{O}(n^{1-t})$	$\tilde{O}(n^{(2d-1)t+1})$
$d \geq 1$ , $\Delta$ -additive approx.	$\tilde{O}(n)$	$\tilde{O}\left(\frac{1}{\Delta^2}\right)$	$\tilde{O}(n)$
$d \geq 1$ , $(1 + \varepsilon)$ -multiplicative approx.	$\tilde{O}(n)$	$\tilde{O}\left(\frac{1}{\varepsilon^2}\right)$	$\tilde{O}(n)$
$d = 1$ , $\varepsilon$ -additive and $(1 + \varepsilon)$ -multiplicative approx.	$\tilde{O}\left(\frac{n}{\varepsilon}\right)$	$\tilde{O}(1)$	$\tilde{O}\left(\frac{n}{\varepsilon}\right)$

- In Section 3, we reduce the set intersection problem to the range entropy query problem in  $\mathbb{R}^2$ . We prove a conditional lower bound showing that we cannot hope for  $O(n \text{ polylog } n)$  space and  $O(\text{polylog } n)$  query time data structures for the range entropy queries.
- Exact data structure for  $d = 1$ . In Section 4.1, we efficiently partition the input points with respect to their  $x$  coordinates into buckets, where each bucket contains a bounded number of points. Given a query interval  $R$ , we visit the bounded number of points in buckets that are partially intersected by  $R$  and we update the overall entropy of the buckets that lie completely inside  $R$ . For any parameter  $t$  chosen by the user, we construct a data structure in  $O(n^{2-t})$  time, with  $O(n^{2(1-t)})$  space and  $O(n^t \log n)$  query time.
- In Section 4.2, instead of partitioning the points with respect to their geometric location, we partition the input points with respect to their colors. We construct  $O(n^{1-t})$  blocks where two sequential blocks contain at most one color in common. Given a query rectangle we visit all blocks and we carefully update the overall entropy. For any tune parameter  $t$  chosen by the user, we construct a data structure in  $O(n \log^{2d} n + n^{(2d-1)t+1} \log^{d+1} n)$  time with  $O(n \log^{2d-1} n + n^{(2d-1)t+1})$  space and  $O(n^{1-t} \log^{2d} n)$  query time.
- Additive approximation. In Subsection 5.1 we use known results for estimating the entropy of an unknown distribution by sampling in the *dual access model*. We propose efficient data structures that apply sampling in a query range in the dual access model. We construct a data structure in  $O(n \log^d n)$  time, with  $O(n \log^{d-1} n)$  space and  $O\left(\frac{\log^{d+3} n}{\Delta^2}\right)$  query time. The data structure returns an additive  $\Delta$ -approximation of the entropy with high probability. It also supports dynamic updates in  $O(\log^d n)$  time.
- Multiplicative approximation. In Subsection 5.2 we propose a multiplicative approximation of the entropy using the results for estimating the entropy in a streaming setting. One significant difference with the previous result is that in information theory at least  $\Omega\left(\frac{\log n}{\varepsilon^2 \cdot H'}\right)$  sampling operations are needed to find get an  $(1 + \varepsilon)$ -multiplicative approximation, where  $H'$  is a lower bound of the entropy. Even if we have efficient data structures for sampling (as we have in additive approximation) we still do not have an efficient query time if the real entropy  $H$  is extremely small. We overcome this technical issue by considering two cases: i) there is no color with total weight more than  $2/3$ , and ii) there exists a color with total weight at most  $2/3$ . While in the latter case the entropy can be extremely small, an additive approximation is sufficient in order to get a multiplicative approximation. In the former one, the entropy is large so we apply the standard sampling method to get a multiplicative approximation. We construct a data structure in

$O(n \log^d n)$  time, with  $O(n \log^d n)$  space and  $O\left(\frac{\log^{d+3}}{\varepsilon^2}\right)$  query time. The data structure returns a multiplicative  $(1 + \varepsilon)$ -approximation of the entropy. It also supports dynamic updates in  $O(\log^d n)$  time.

- Additive and multiplicative approximation. In Subsection 5.3, we propose a new data structure for approximating the entropy in the query range for  $d = 1$ . We get the intuition from data structures counting the number of colors in a query interval. Such a data structure finds a geometric mapping to a different geometric space, such that if at least a point with color  $u_i$  exists in the original  $P \cap R$ , then there is a unique point with color  $u_i$  in the corresponding query range in the new geometric space. Unfortunately, this property is not sufficient for finding the entropy. Instead, we need to know more information about the weights of the points and the entropy in canonical subsets of the new geometric space, which is challenging to do. We construct a data structure in  $O\left(\frac{n}{\varepsilon} \log^5 n\right)$  time, with  $O\left(\frac{n}{\varepsilon} \log^2 n\right)$  space and  $O\left(\log^2 n \log \frac{\log n}{\varepsilon}\right)$  query time. The data structure returns an  $(1 + \varepsilon)$ -multiplicative and  $\varepsilon$ -additive approximation of the entropy.
- Partitioning using entropy. In Section 6 we show how our new data structures can be used to run partitioning algorithms over time series, histograms, and points efficiently.

**Related work.** Entropy has been used a lot for partitioning to create histograms in databases. For example, To et al. [38] used entropy to design histograms for selectivity estimation queries. In particular, they aim to find a partitioning of  $k$  buckets in 1d such that the cumulative entropy is maximized. They consider a special case where they already have a histogram (so all items of the same color are accumulated to the same location) and the goal is to partition the histogram into  $k$  buckets. They propose a greedy algorithm that finds a local optimum solution. However there is no guarantee on the overall optimum partitioning. Using our new data structures, we can find the entropy in arbitrary range queries, which is not supported in [38]. Our data structures can also be used to accelerate partitioning algorithms with theoretical guarantees (see Subsection 6) in a more general setting, where points of the same color have different locations.

In addition, there is a number of papers that use entropy to find a clustering of items. Cruz et al. [19] used entropy for the community detection problem in augmented social networks. They describe a greedy algorithm that exchanges two random nodes between two random clusters if the entropy of the new instance is lower. Barbará et al. [6] used the *expected entropy* for categorical clustering. They describe a greedy algorithm that starts with a set of initial clusters, and for each new item decides to place it in the cluster that has the lowest entropy. Li et al. [29] also used the expected entropy for categorical clustering but they extend it to probabilistic clustering models. Finally, Ben-Gal et al. [8] used the expected entropy to develop an entropy-based clustering measure that measures the homogeneity of mobility patterns within clusters of users. All these methods do not study the problem of finding the entropy in a query range efficiently. While these methods perform well in practice, it is challenging to derive theoretical guarantees. In spatial databases items are represented as points in  $\mathbb{R}^d$ , so our new data structures could be used to find faster and better entropy-based clustering techniques. For example, we could run range entropy queries with different radii around a center until we find a cluster with small radius and small (or large) expected entropy.

There is a lot of work on computing an approximation of the entropy in the streaming setting [11, 15, 23, 28]. For a stream of  $m$  distinct values ( $m$  colors in our setting) Chakrabarti et al. [14] compute an  $(1 + \varepsilon)$ -multiplicative approximation of the entropy in a single pass using  $O(\varepsilon^{-2} \log(\delta^{-1}) \log m)$  words of space, with probability at least  $1 - \delta$ . For a stream

of size  $n$  ( $n$  points in our setting) Clifford and Cosma [17] propose a single-pass  $\varepsilon$ -additive algorithm using  $O(\varepsilon^{-2} \log n \log(n\varepsilon^{-1}))$  bits with bounded probability. Harvey et al. [25] allow deletions in the streaming setting and they propose a single-pass  $(1 + \varepsilon)$ -multiplicative algorithm using  $\tilde{O}(\varepsilon^{-2} \log^2 m)$  words of space with bounded probability. Furthermore, they propose a single-pass  $\varepsilon$ -additive approximation using  $\tilde{O}(\varepsilon^{-2} \log m)$  words of space. While some techniques from the streaming setting are useful in our query setting, the two problems are fundamentally different. In the streaming setting, preprocessing is not allowed, all data are processed one by one and an estimation of the entropy is maintained. In our setting, the goal is to construct a data structure such that given any query range, the entropy of the items in the range should be computed in sublinear time, i.e., without processing all items in the query range during the query phase.

Let  $\mathcal{D}$  be an unknown discrete distribution over  $n$  values. There is an interesting line of work on approximating the entropy of  $\mathcal{D}$  by sampling in the *dual access model*. Batu et al. [7] give an  $(1 + \varepsilon)$ -multiplicative approximation of the entropy of  $\mathcal{D}$  with sample complexity  $O(\frac{(1+\varepsilon)^2 \log^2 n}{\varepsilon^2 \cdot H'})$ , where  $H'$  is a lower bound of the actual entropy  $H(\mathcal{D})$ . Guha et al. [23] improved the sample complexity to  $O(\frac{\log n}{\varepsilon^2 \cdot H'})$ , matching the lower bound  $\Omega(\frac{\log n}{(2+\varepsilon)\varepsilon^2 \cdot H'})$  found in [7]. Canonne and Rubinfeld [13] describe a  $\Delta$ -additive approximation of the entropy with sample complexity  $O(\frac{\log^2 n}{\Delta^2})$ . Cafarov et al. [12] show that  $\Omega(\frac{\log^2 n}{\Delta^2})$  sample queries are necessary to get  $\Delta$ -additive approximation. All these algorithms return the correct approximations with constant probability. If we want to guarantee the result with high probability then the sample complexity is multiplied by a  $\log n$  factor.

A related query to estimating the entropy is the range color query. Given a set of colored points in  $\mathbb{R}^d$ , the goal is to construct a data structure such that given a query rectangle, it returns the number of colors in the query range.

## 2 Preliminaries

Let  $P$  be a set of  $n$  points in  $\mathbb{R}^d$  and let  $U$  be a set of  $m$  colors  $U = \{u_1, \dots, u_m\}$ . Each point  $p \in P$  is associated with a color from  $U$ , i.e.,  $u(p) = u_i$  for  $u_i \in U$ . Furthermore, each point  $p \in P$  is associated with a non-negative weight  $w(p) \geq 0$ . For a subset of points  $P' \subseteq P$ , let  $P'(u_i) = \{p \in P' \mid u(p) = u_i\}$ , for  $i \leq m$ , be the set of points having color  $u_i$ . Let  $u(P') = \{u_i \mid \exists p \in P', u(p) = u_i\}$  be the set of colors of the points in  $P'$ . Finally, let  $w(P') = \sum_{p \in P'} w(p)$ . The entropy of set  $P'$  is defined as  $H(P') = \sum_{i=1}^m \frac{w(P'(u_i))}{w(P')} \log \left( \frac{w(P')}{w(P'(u_i))} \right)$ .

For simplicity, and without loss of generality, we can consider throughout the paper that  $w(p) = 1$  for each point  $p \in P$ . All the results, proofs, and properties we show hold for the weighted case almost verbatim. Hence, from now on, we assume  $w(p) = 1$  and the definition of entropy becomes

$$H(P') = \sum_{i=1}^m \frac{|P'(u_i)|}{|P'|} \log \left( \frac{|P'|}{|P'(u_i)|} \right) = \sum_{u_i \in u(P')} \frac{|P'(u_i)|}{|P'|} \log \left( \frac{|P'|}{|P'(u_i)|} \right). \quad (1)$$

If  $|P'(u_i)| = 0$ , then we consider that  $\frac{|P'(u_i)|}{|P'|} \log \left( \frac{|P'|}{|P'(u_i)|} \right) = 0$ .

**Updating the entropy.** Let  $P_1, P_2 \subset P$  be two subsets of  $P$  such that  $u(P_1) \cap u(P_2) = \emptyset$ . The next formula for the entropy of  $P_1 \cup P_2$  is known (see [38])

$$H(P_1 \cup P_2) = \frac{|P_1|H(P_1) + |P_2|H(P_2) + |P_1| \log \left( \frac{|P_1| + |P_2|}{|P_1|} \right) + |P_2| \log \left( \frac{|P_1| + |P_2|}{|P_2|} \right)}{|P_1| + |P_2|}. \quad (2)$$



## 6:6 Range Entropy Queries and Partitioning

If  $|u(P_2)| = 1$  then

$$H(P_1 \cup P_2) = \frac{|P_1|H(P_1)}{|P_1| + |P_2|} + \frac{|P_1|}{|P_1| + |P_2|} \log \left( \frac{|P_1| + |P_2|}{|P_1|} \right) + \frac{|P_2|}{|P_1| + |P_2|} \log \left( \frac{|P_1| + |P_2|}{|P_2|} \right). \quad (3)$$

Finally, if  $P_3 \subset P_1$  with  $|u(P_3)| = 1$  and  $u(P_1 \setminus P_3) \cap u(P_3) = \emptyset$  then

$$H(P_1 \setminus P_3) = \frac{|P_1|}{|P_1| - |P_3|} \left( H(P_1) - \frac{|P_3|}{|P_1|} \log \frac{|P_1|}{|P_3|} - \frac{|P_1| - |P_3|}{|P_1|} \log \frac{|P_1|}{|P_1| - |P_3|} \right). \quad (4)$$

We notice that in all cases, if we know  $H(P_1), H(P_2)$  and the cardinality of each subset we can update the entropy in  $O(1)$  time.

**Range queries.** In some data structures we need to handle range reporting or range counting problems. Given  $P$ , we need to construct a data structure such that given a query rectangle  $R$ , the goal is to return  $|R \cap P|$ , or report  $R \cap P$ . We use range trees [10]. A range tree can be constructed in  $O(n \log^d n)$  time, it has  $O(n \log^{d-1} n)$  space and can answer an aggregation query (such as count, sum, max etc.) in  $O(\log^d n)$  time. A range tree can be used to report  $R \cap P$  in  $O(\log^d n + |R \cap P|)$  time. Using *fractional cascading* the  $\log^d n$  term can be improved to  $\log^{d-1} n$  in the query time. However, for simplicity, we consider the simple version of a range tree without using fractional cascading. Furthermore, a range tree can be used to return a uniform sample point from  $R \cap P$  in  $O(\log^d n)$  time. We give more details about range trees and sampling in the full version of the paper [27]. There is also lot of work on designing data structures for returning  $k$  independent samples in a query range efficiently [1, 2, 26, 32, 37, 39, 40]. For example, if the input is a set of points in  $\mathbb{R}^d$  and the query range is a query hyper-rectangle, then there exists a data structure [32] with space  $O(n \log^{d-1} n)$  and query time  $O(\log^d n + k \log n)$ . For our purposes, it is sufficient to run  $k$  independent sampling queries in a (modified) range tree with total query time  $O(k \log^d n)$ .

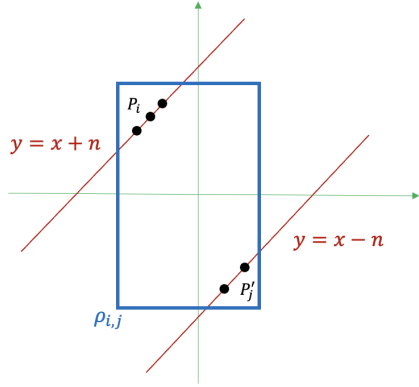
**Expected entropy and monotonicity.** Entropy is not monotone because if  $P_1 \subseteq P_2$ , it does not always hold that  $H(P_1) \leq H(P_2)$ . Using the results in [29], we can show that  $H(P_1) \geq \frac{|P_1|-1}{|P_1|} H(P_1 \setminus \{p\})$ , for a point  $p \in P_1 \subset P$ . If we multiply with  $|P_1|/n$  we have  $\frac{|P_1|}{n} H(P_1) \geq \frac{|P_1|-1}{n} H(P_1 \setminus \{p\})$ . Hence, we show that, for  $P_1 \subseteq P_2 \subseteq P$ ,  $\frac{|P_1|}{n} H(P_1) \leq \frac{|P_2|}{n} H(P_2)$ . The quantity  $\frac{|P_1|}{|P_1|} H(P_1)$  is called expected entropy. This monotonicity property helps us to design efficient partitioning algorithms with respect to expected entropy, for example, find a partitioning that minimizes the cumulative or maximum expected entropy.  $\quad \text{t}$

### 3 Lower Bound

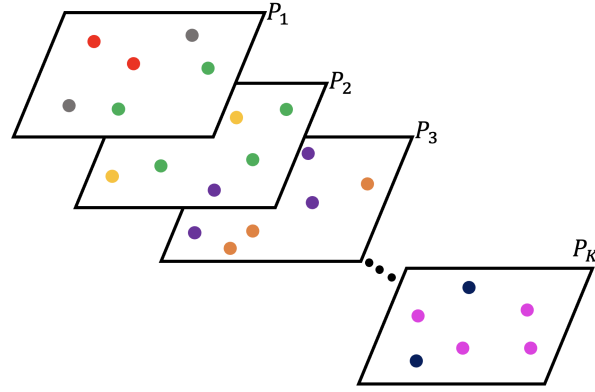
In this section, we give a lower bound for range entropy queries in the real-RAM model. We show a reduction from the set intersection problem that suggests that data structures with near-linear space and polylogarithmic query time are unlikely to exist even for  $d = 2$ .

The set intersection problem is defined as follows. Given a family of sets  $S_1, \dots, S_g$ , with  $\sum_{i=1}^g |S_i| = n$ , the goal is to construct a data structure such that given a query pair of indices  $i, j$ , the goal is to decide if  $S_i \cap S_j = \emptyset$ . It is widely believed that for any positive value  $Q \in \mathbb{R}$ , any data structure for the set intersection problem with  $O(Q)$  query time needs  $\Omega \left( \left( \frac{n}{Q} \right)^2 \right)$  space [20, 35, 36], skipping  $\log^{O(1)} n$  factors. Next, we show that any data structure for solving the range entropy query can be used to solve the set intersection problem.





■ **Figure 1** Lower bound construction.



■ **Figure 2** Partition  $P$  into  $K$  buckets in  $\mathbb{R}^2$ . Two consecutive buckets have at most one color in common.

Let  $S_1, \dots, S_g$  be an instance of the set intersection problem as we defined above. We design an instance of the range entropy query constructing a set  $P$  of  $2n$  points in  $\mathbb{R}^2$  and  $|U| = |\bigcup_i S_i|$ . Let  $n_0 = 0$  and  $n_i = n_{i-1} + |S_i|$  for  $i = 1, \dots, g$ . Let  $s_{i,k}$  be the value of the  $k$ -th item in  $S_i$  (we consider any arbitrary order of the items in each  $S_i$ ). Let  $S = \bigcup_i S_i$ , and  $q = |S|$ . Let  $\sigma_1, \dots, \sigma_q$  be an arbitrary ordering of  $S$ . We set  $U = \{1, \dots, q\}$ . Next, we create a geometric instance of  $P$  in  $\mathbb{R}^2$ : All points lie on two parallel lines  $L = x + n$ , and  $L' = x - n$ . For each  $s_{i,k}$  we add in  $P$  two points,  $p_{i,k} = (-(k + n_{i-1}), -(k + n_{i-1}) + n)$  on  $L$ , and  $p'_{i,k} = ((k + n_{i-1}), k + n_{i-1} - n)$  on  $L'$ . If  $s_{i,k} = \sigma_j$  for some  $j \leq q$ , we set the color/category of both points  $p_{i,k}, p'_{i,k}$  to be  $j$ . Let  $P_i$  be the set of points corresponding to  $S_i$  that lie on  $L$ , and  $P'_i$  the set of points corresponding to  $S_i$  that lie on  $L'$ . We set  $P = \bigcup_i (P_i \cup P'_i)$ . We note that for any pair  $i, j$ , points  $P_i \cup P'_j$  have distinct categories if and only if  $S_i \cap S_j = \emptyset$ .  $P$  uses  $O(n)$  space and can be constructed in  $O(n)$  time.

Let  $\mathcal{D}$  be a data structure for range entropy queries with space  $S(n)$  and query time  $Q(n)$  constructed on  $n$  points. Given an instance of the set intersection problem, we construct  $P$  as described above. Then we build  $\mathcal{D}$  on  $P$  and we construct a range tree  $\mathcal{T}$  on  $P$  for range counting queries. Given a pair of indexes  $i, j$  the question is if  $S_i \cap S_j = \emptyset$ . We answer this question using  $\mathcal{D}$  and  $\mathcal{T}$  on  $P$ . Geometrically, it is known we can find a rectangle  $\rho_{i,j}$  in  $O(1)$  time such that  $\rho_{i,j} \cap P = P_i \cup P'_j$  (see Figure 1). We run the range entropy query  $\mathcal{D}(\rho_{i,j})$  and the range counting query  $\mathcal{T}(\rho_{i,j})$ . Let  $H_{i,j}$  be the entropy of  $P_i \cup P'_j$  and  $n_{i,j} = |P_i \cup P'_j|$ . If  $H_{i,j} = \log n_{i,j}$  we return that  $S_i \cap S_j = \emptyset$ . Otherwise, we return  $S_i \cap S_j \neq \emptyset$ .

The data structure we construct for answering the set intersection problem has  $O(S(2n) + n \log n) = \tilde{O}(S(2n))$  space. The query time is  $(Q(2n) + \log n)$  or just  $O(Q(n))$  assuming that  $Q(n) \geq \log n$ .

► **Lemma 5.** *In the preceding reduction,  $S_i \cap S_j = \emptyset$  if and only if  $H_{i,j} = \log n_{i,j}$ .*

**Proof.** If  $S_i \cap S_j = \emptyset$  then from the construction of  $P$  we have that all colors in  $P_i \cup P'_j$  are distinct, so  $n_{i,j} = |u(P_i \cup P'_j)|$ . Hence, the entropy  $H(P_i \cup P'_j)$  takes the maximum possible value which is  $H(P_i \cup P'_j) = \sum_{v \in u(P_i \cup P'_j)} \frac{1}{n_{i,j}} \log n_{i,j} = \log n_{i,j}$ .

If  $H_{i,j} \neq \log n_{i,j}$  we show that  $S_i \cap S_j \neq \emptyset$ . The maximum value that  $H_{i,j}$  can take is  $\log n_{i,j}$  so we have  $H_{i,j} < \log n_{i,j}$ . The entropy is a measure of uncertainty of a distribution. It is known that the discrete distribution with the maximum entropy is unique and it is the uniform distribution. Any other discrete distribution has entropy less than  $\log n_{i,j}$ . Hence the result follows. ◀

We conclude with the next theorem.

► **Theorem 6.** *If there is a data structure for range entropy queries with  $S(n)$  space and  $Q(n)$  query time, then for the set intersection problem there exists a data structure with  $O(S(2n))$  space and  $O(Q(2n))$  query time, skipping  $\log n$  factors.*

## 4 Exact Data Structures

In this section we describe data structures that return the entropy in a query range, exactly. First, we provide a data structure for  $d = 1$  and we extend it to any constant dimension  $d$ . Next, we provide a second data structure for any constant dimension  $d$ . The first data structure is better for  $d = 1$ , while the second data structure is better for any constant  $d > 1$ .

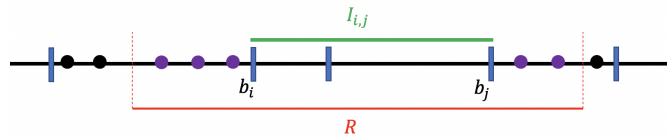
### 4.1 Efficient data structure for $d = 1$

Let  $P$  be a set of  $n$  points in  $\mathbb{R}^1$ . Since the range entropy query problem is not decomposable, the main idea is to precompute the entropy in some carefully chosen canonical subsets of  $P$ . When we get a query interval  $R$ , we find the maximal precomputed canonical subset in  $R$ , and then for each color among the colors of points in  $R$  not included in the canonical subset, we update the overall entropy using Equations 2, 3, and 4. We also describe how we can precompute the entropy of all canonical subsets efficiently.

**Data Structure.** Let  $t \in [0, 1]$  be a parameter. Let  $B_t = \{b_1, \dots, b_k\}$  be  $k = n^{1-t}$  points in  $\mathbb{R}^1$  such that  $|P \cap [b_j, b_{j+1}]| = n^t$ , for any  $j < n^{1-t}$ . For any pair  $b_i, b_j \in B_t$  let  $I_{i,j} = [b_i, b_j]$  be the interval with endpoints  $b_i, b_j$  and let  $I$  be the set of all intervals. For any pair  $b_i, b_j$  we store the interval  $I_{i,j}$  and we precompute  $H_{i,j} = H(P \cap I_{i,j})$ , and  $n_{i,j} = |P \cap I_{i,j}|$ . Next, we construct an interval tree  $\mathcal{T}$  on  $I$ . Finally, for each color  $u \in u(P)$  we construct a search binary tree  $\mathcal{T}_u$  over  $P(u)$ .

We have  $|B_t| = O(n^{1-t})$  so  $|I| = O(n^{2(1-t)})$ . The interval tree along with all the search binary trees have  $O(n)$  space in total. Hence we need  $O(n^{2(1-t)})$  space for our data structure. In the full version [27], we show how we can construct the data structure in  $O(n^{2-t})$  time.

**Query procedure.** Given a query interval  $R$ , we find the maximal interval  $I_{i,j} \in I$  such that  $I \subseteq R$  using the interval tree. Recall that we have precomputed the entropy  $H_{i,j}$ . Let  $H = H_{i,j}$  be a variable that we will update throughout the algorithm storing the current entropy. Let also  $N = n_{i,j}$  be the variable that stores the number of items we currently consider to compute  $H$ . Let  $P_R = P \cap (R \setminus I_{i,j})$  be the points in  $P \cap R$  that are not included in the maximal interval  $I_{i,j}$ . See also Figure 3.



■ **Figure 3** Instance of the query algorithm given query interval  $R$ . Purple points are points in  $P_R$ .

We visit each point in  $P_R$  and we identify  $u(P_R)$ . For each  $\mathbf{u} \in u(P_R)$ , we run a query in  $\mathcal{T}_{\mathbf{u}}$  with range  $I_{i,j}$  finding the number of points in  $P \cap I_{i,j}$  with color  $\mathbf{u}$ . Let  $n_{\mathbf{u}}$  be this count.

If  $n_{\mathbf{u}} = 0$  then there is no point in  $P \cap I_{i,j}$  with color  $\mathbf{u}$  so we insert  $|P_R(\mathbf{u})|$  items of color  $\mathbf{u}$  in the current entropy using Equation 3. In that formula,  $|P_1| = N$ ,  $H(P_1) = H$  and  $|P_2| = |u(P_R)|$ . We update  $N = N + |u(P_R)|$ , and  $H$  with the updated entropy  $H(P_1 \cup P_2)$ .

If  $n_{\mathbf{u}} > 0$  then there is at least one point in  $P \cap I_{i,j}$  with color  $\mathbf{u}$ . Hence, we update the entropy  $H$ , by first removing the  $n_{\mathbf{u}}$  points of color  $\mathbf{u}$  in  $P \cap I_{i,j}$  and then re-inserting  $n_{\mathbf{u}} + |u(P_R)|$  points of color  $\mathbf{u}$ . We use Equation 4 for removing the points with color  $\mathbf{u}$  with  $|P_1| = N$ ,  $H(P_1) = H$ , and  $|P_3| = n_{\mathbf{u}}$ . We update  $N = N - n_{\mathbf{u}}$  and  $H$  with the updated entropy  $H(P_1 \setminus P_3)$ . Then we use Equation 3 for re-inserting the points with color  $u$ , with  $|P_1| = N$ ,  $H(P_1) = H$ , and  $|P_2| = n_{\mathbf{u}} + |u(P_R)|$ . We update  $N = N + n_{\mathbf{u}} + |u(P_R)|$  and  $H$  with the updated entropy  $H(P_1 \cup P_2)$ . After visiting all colors in  $u(P_R)$ , we return the updated entropy  $H$ . The correctness of the algorithm follows from Equations 3, 4. For each color  $u \in u(P_R)$  we update the entropy including all points of color  $u$ .

For a query interval  $R$  we run a query in the interval tree to find  $I_{i,j}$  in  $O(\log n)$  time. The endpoints of  $R$  intersect two intervals  $[b_h, b_{h+1}]$  and  $[b_v, b_{v+1}]$ . Recall that by definition, such interval contains  $O(n^t)$  points from  $P$ . Hence,  $|P_R| = O(n^t)$  and  $|u(P_R)| = O(n^t)$ . For each  $\mathbf{u} \in u(P_R)$ , we spend  $O(\log n)$  time to search  $\mathcal{T}_{\mathbf{u}}$  and find  $n_{\mathbf{u}}$ . Then we update the entropy in  $O(1)$  time. Overall, the query procedure takes  $O(n^t \log n)$  time.

► **Theorem 7.** *Let  $P$  be a set of  $n$  points in  $\mathbb{R}^1$ , where each point is associated with a color, and let  $t \in [0, 1]$  be a parameter. A data structure of  $O(n^{2(1-t)})$  size can be computed in  $O(n^{2-t})$  time, such that given a query interval  $R$ ,  $H(P \cap R)$  can be computed in  $O(n^t \log n)$  time.*

In the full version [27], we extend this data structure to any constant  $d > 1$ .

## 4.2 Efficient data structure for $d > 1$

While the previous data structure can be extended to higher dimensions, here we propose a more efficient data structure for  $d > 1$ . In this data structure we split the points with respect to their colors. The data structure has some similarities with the data structure presented in [3, 4] for the max query under uncertainty, however the two problems are different and there are key differences on the way we construct the data structure and the way we compute the result of the query.

**Data Structure.** We first consider an arbitrary permutation of the colors in  $U$ , i.e.  $u_1, \dots, u_m$ . The order used to partition the items is induced from the permutation over the colors. Without loss of generality we set  $u_j = j$  for each  $j \leq m$ . We split  $P$  into  $K = O(n^{1-t})$  buckets  $P_1, \dots, P_K$  such that i) each bucket contains  $O(n^t)$  points, and ii) for every point  $p \in P_i$  and  $q \in P_{i+1}$ ,  $u(p) \geq u(q)$ . We notice that for any pair of buckets  $P_i, P_{i+1}$  it holds  $|u(P_i) \cap u(P_{i+1})| \leq 1$ , see Figure 2. We slightly abuse the notation and we use  $P_i$  to represent both the  $i$ -th bucket and the set of points in the  $i$ -th bucket.

For each bucket  $P_i$ , we take all combinatorially different (hyper)rectangles  $R_i$  defined by the points  $P_i$ . For each such rectangle  $r$ , we precompute and store the entropy  $H(P_i \cap r)$  along with the number of points  $n(P_i \cap r) = |P_i \cap r|$ . In addition, we store  $u^+(r)$ , the color with the maximum value (with respect to the permutation of the colors) in  $r \cap P_i$ . Furthermore, we store  $u^-(r)$ , the color with the minimum value in  $r \cap P_i$ . Let  $n^+(r) = |\{p \in r \cap P_i \mid u(p) = u^+(r)\}|$  and  $n^-(r) = |\{p \in r \cap P_i \mid u(p) = u^-(r)\}|$ . Finally, for each bucket  $P_i$  we construct a modified range tree  $\mathcal{T}'_i$  over all  $R_i$ , such that given a query rectangle  $R$  it returns the maximal rectangle  $r \in R_i$  that lies completely inside  $R$ . We note that  $r \cap P_i = R \cap P_i$ . This can be done by representing the  $d$ -dimensional hyper-rectangles as  $2d$ -dimensional points merging the coordinates of two of their corners.

## 6:10 Range Entropy Queries and Partitioning

Overall, we need  $O(n \log^{2d-1} n)$  space for the modified range trees  $\mathcal{T}'_i$ , and  $O(n^{1-t} \cdot n^{2dt}) = O(n^{(2d-1)t+1})$  space to store all additional information (entropy, counts, max/min color) in each rectangle. This is because there are  $O(n^{1-t})$  buckets, and in each bucket there are  $O(n^{2dt})$  combinatorially different rectangles. Overall, our data structure has  $O\left(n \log^{2d-1} n + n^{(2d-1)t+1}\right)$  space.

**Query Procedure.** We are given a query (hyper)rectangle  $R$ . We visit the buckets  $P_1, \dots, P_K$  in order and compute the entropy for  $R \cap (P_1 \cup \dots \cup P_i)$ . Let  $H$  be the overall entropy we have computed so far. For each bucket  $P_i$  we do the following: First we run a query using  $\mathcal{T}'_i$  to find  $r_i \in R_i$  that lies completely inside  $R$ . Then we update the entropy  $H$  considering the items in  $P_i \cap r_i$ . If  $u^-(r_{i-1}) = u^+(r_i)$  then we update the entropy  $H$  by removing  $n^-(r_{i-1})$  points with color  $u^-(r_{i-1})$  using Equation 4. Then we insert  $n^-(r_{i-1}) + n^+(r_i)$  points of color  $u^+(r_i)$  in  $H$  using Equation 3. Finally, we remove  $n^+(r_i)$  points of color  $u^+(r_i)$  from the precomputed  $H(P_i \cap r_i)$  using Equation 4 and we merge the updated  $H$  with  $H(P_i \cap r_i)$  using Equation 2. We note that in the last step we can merge the updated  $H$  with the updated  $H(P_i \cap r_i)$  because no color from the points used to compute the current  $H$  is appeared in the points used to compute the current  $H(P_i \cap r_i)$ . On the other hand, if  $u^-(r_{i-1}) \neq u^+(r_i)$ , then we merge the entropies  $H$  and  $H(P_i \cap r_i)$  using directly Equation 2.

In each bucket  $P_i$  we need  $O(\log^{2d} n)$  to identify the maximal rectangle  $r_i$  inside  $R$ . Then we need  $O(1)$  time to update the current entropy  $H$ . Overall, we need  $O(n^{1-t} \log^{2d} n)$  time.

**Fast Construction.** All range trees can be computed in  $O(n \log^{2d} n)$  time. Next, we focus on computing  $H(P_i \cap r)$  for all rectangles  $r \in R_i$ . We compute the other quantities  $n(P_i \cap r)$ ,  $u^-(r)$ , and  $u^+(r)$  with a similar way. A straightforward way is to consider every possible rectangle  $r$  and compute independently the entropy in linear time. There are  $O(n^{2dt})$  rectangles so the running time is  $O(n^{2dt+1})$ . We propose a faster construction algorithm.

The main idea is to compute the entropy for rectangles in a specific order. In particular, we compute the entropy of rectangles that contain  $c$  points after we compute the entropies for rectangles that contain  $c - 1$  points. Then we use Equations 3, 4 to update the entropy of the new rectangle without computing it from scratch. Overall, we construct the data structure in  $O(n^{(2d-1)t+1} \log^{d+1} n)$  time. We describe the missing details in the full version of the paper [27].

► **Theorem 8.** *Let  $P$  be a set of  $n$  points in  $\mathbb{R}^d$ , where each point is associated with a color, and let  $t \in [0, 1]$  be a parameter. A data structure of  $O(n \log^{2d-1} n + n^{(2d-1)t+1})$  size can be computed in  $O(n \log^{2d} n + n^{(2d-1)t+1} \log^{d+1} n)$  time, such that given a query hyper-rectangle  $R$ ,  $H(P \cap R)$  can be computed in  $O(n^{1-t} \log^{2d} n)$  time.*

## 5 Approximate Data Structures

In this section we describe data structures that return the entropy in a query range, approximately. First, we provide a data structure that returns an additive approximation of the entropy and next we provide a data structure that returns a multiplicative approximation efficiently. Finally, for  $d = 1$ , we design a deterministic and more efficient data structure that returns an additive and multiplicative approximation of the entropy.

## 5.1 Additive approximation

In this Subsection, we construct a data structure on  $P$  such that given a query rectangle  $R$  and a parameter  $\Delta$ , it returns a value  $h$  such that  $H(P \cap R) - \Delta \leq h \leq H(P \cap R) + \Delta$ . The intuition comes from the area of finding an additive approximation of the the entropy of an unknown distribution in the dual access model [13].

Let  $D$  be a fixed distribution over a set of values  $\alpha_1, \dots, \alpha_N$ . Each value  $\alpha_i$  has a probability  $D(\alpha_i)$  which is not known, such that  $\sum_{i=1}^N D(\alpha_i) = 1$ . The authors in [13] show that if we ask  $O\left(\frac{\log^2 \frac{N}{\Delta} \log N}{\Delta^2}\right)$  sample queries in the dual access model, then we can get a  $\Delta$  additive-approximation of the entropy of  $D$  with high probability in  $O\left(\frac{\log^2 \frac{N}{\Delta} \log N}{\Delta^2} \mathcal{S}\right)$  time, where  $\mathcal{S}$  is the running time to get a sample. In the dual access model, we consider that we have a dual oracle for  $D$  which is a pair of oracles ( $\text{SAMP}_D, \text{EVAL}_D$ ). When required, the sampling oracle  $\text{SAMP}_D$  returns a value  $\alpha_i$  with probability  $D(\alpha_i)$ , independently of all previous calls to any oracle. Furthermore, the evaluation oracle  $\text{EVAL}_D$  takes as input a query element  $\alpha_i$  and returns the probability weight  $D(\alpha_i)$ .

Next, we describe how the result above can be used in our setting. The goal in our setting is to find the entropy  $H(P')$ , where  $P' = P \cap R$ , for a query rectangle  $R$ . The colors in  $u(P')$  define the distinct values in distribution  $D$ . By definition, the number of colors is bounded by  $|P'| = O(n)$ . The probability weight is defined as  $\frac{|P'(u_i)|}{|P'|}$ . We note that in [13] they assume that they know  $N$ , i.e., the number of values in distribution  $D$ . In our case, we cannot compute the number of colors  $|u(P')|$  efficiently. Even though we can easily compute an  $O(\log^d n)$  approximation of  $|u(P')|$ , it is sufficient to use the loose upper bound  $|u(P')| \leq n$ . This is because, without loss of generality, we can assume that there exist  $n - |u(P')|$  values/colors with probability (arbitrarily close to) 0. All the results still hold. Next we present our data structure to simulate the dual oracle.

**Data structure.** For each color  $u_i \in U$  we construct a range tree  $\mathcal{T}_i$  on  $P(u_i)$  for range counting queries. We also construct another range tree  $\mathcal{T}$  on  $P$  for range counting queries, which is independent of the color. Next, we construct a range tree  $\mathcal{S}$  on  $P$  for range sampling queries. In particular, by pre-computing the number of points stored in the subtree of each node of the range tree, we can return a sample in a query region efficiently. For more details the reader can check the full version of the paper [27], and [1, 2, 26, 32, 37, 39, 40] where the authors propose a data structure for finding  $k$  samples in a query region efficiently<sup>1</sup>. We need  $O(n \log^d n)$  time to construct all the range trees, while the overall space is  $O(n \log^{d-1} n)$ .

**Query procedure.** The query procedure involves the algorithm for estimating the entropy of an unknown distribution in the dual access model [13]. Here, we only need to describe how to execute the oracles  $\text{SAMP}_D$  and  $\text{EVAL}_D$  in  $P' = P \cap R$  using the data structure.

- **SAMP<sub>D</sub>:** Recall that  $\text{SAMP}_D$  returns  $\alpha_i$  with probability  $D(\alpha_i)$ . In our setting, values  $\alpha_1, \dots, \alpha_n$  correspond to colors. So, the goal is to return a color  $u_i$  with probability proportional to the number of points with color  $u_i$  in  $P'$ . Indeed,  $\mathcal{S}$  returns a point  $p$  uniformly at random in  $P'$ . Hence, the probability that a point with color  $u_i$  is found is  $\frac{|P'(u_i)|}{|P'|}$ .

<sup>1</sup> While it is known how to get  $k$  independent weighted samples in a query hyper-rectangle in  $O(\log^d n + k \log n)$  time [32], the overall asymptotic query time of our problem remains the same if we use a range tree as described in [27] with  $O(k \log^d n)$  query time.

■ **EVAL<sub>D</sub>**: Recall that given a value  $\alpha_i$ , **EVAL<sub>D</sub>** returns the probability weight  $D(\alpha_i)$ . Equivalently, in our setting, given a color  $u_i$ , the goal is to return  $\frac{|P'(u_i)|}{|P'|}$ . Using  $\mathcal{T}_i$  we run a counting query in the query rectangle  $R$  and find  $|P'(u_i)|$ . Then using  $\mathcal{T}$ , we run a counting query in  $R$  and we get  $|P'|$ . We divide the two quantities and return the result. In each iteration, every oracle call **SAMP<sub>D</sub>** and **EVAL<sub>D</sub>** executes a constant number of range tree queries, so the running time is  $O(\log^d n)$ . The algorithm presented in [13] calls the oracles  $O\left(\frac{\log^2 \frac{n}{\Delta} \log n}{\Delta^2}\right)$  times to guarantee the result with probability at least  $1 - 1/n$ , so the overall query time is  $O\left(\frac{\log^{d+1} n \cdot \log^2 \frac{n}{\Delta}}{\Delta^2}\right)$ . We note that if  $\Delta < \frac{1}{\sqrt{n}}$  then the query time is  $\Omega(n \log n)$ . However, it is trivial to compute the entropy in  $P \cap R$  in  $O(n \log n)$  time by traversing all points in  $P \cap R$ . Hence, the additive approximation is non-trivial when  $\Delta \geq \frac{1}{\sqrt{n}}$ . In this case,  $\log^2 \frac{n}{\Delta^2} = O(\log^2 n)$ . We conclude that the query time is bounded by  $O\left(\frac{\log^{d+3} n}{\Delta^2}\right)$ . We conclude with the next theorem.

► **Theorem 9.** *Let  $P$  be a set of  $n$  points in  $\mathbb{R}^d$ , where each point is associated with a color. A data structure of  $O(n \log^{d-1} n)$  size can be computed in  $O(n \log^d n)$  time, such that given a query hyper-rectangle  $R$  and a real parameter  $\Delta$ , a value  $h$  can be computed in  $O\left(\frac{\log^{d+3} n}{\Delta^2}\right)$  time, such that  $H(P \cap R) - \Delta \leq h \leq H(P \cap R) + \Delta$ , with high probability.*

This data structure can be made dynamic under arbitrary insertions and deletions of points using well known techniques [9, 21, 33, 34]. The update time is  $O(\log^d n)$ .

## 5.2 Multiplicative approximation

In this Subsection, we construct a data structure such that given a query rectangle  $R$  and a parameter  $\varepsilon$ , it returns a value  $h$  such that  $\frac{1}{1+\varepsilon}H(P \cap R) \leq h \leq (1+\varepsilon)H(P \cap R)$ . The intuition comes from the area of finding a multiplicative approximation of the the entropy of an unknown distribution in the dual access model [23] and the streaming algorithms for finding a multiplicative approximation of the the entropy [14]. In particular, in this section we extend the streaming algorithm proposed in [14] to work in the query setting.

We use the notation from the previous Subsection where  $D$  is an unknown distribution over a set of values  $\alpha_1, \dots, \alpha_N$ . It is known [23] that if we ask  $O\left(\frac{\log N}{\varepsilon^2 \cdot H'}\right)$  queries in the dual access model, where  $H'$  is a lower bound of the actual entropy of  $D$ , i.e.,  $H(D) \geq H'$ , then we can get an  $(1+\varepsilon)$ -multiplicative approximation of the entropy of  $D$  with high probability, in  $O\left(\frac{\log N}{\varepsilon^2 \cdot H'}\mathcal{S}\right)$  time, where  $\mathcal{S}$  is the time to get a sample. We consider that we have a dual oracle for  $D$  which is a pair of oracles (**SAMP<sub>D</sub>**, **EVAL<sub>D</sub>**), as we had in additive approximation. Similarly to the additive approximation, in our setting we do not know the number of colors in  $P' = P \cap R$  or equivalently the number of values  $N$  in distribution  $D$ . However it is sufficient to use the upper bound  $|u(P')| \leq n$  considering  $n - |u(P')|$  colors with probability (arbitrarily close to) 0. If we use the same data structure constructed for the additive approximation, we could solve the multiplicative-approximation, as well. While this is partially true, there is a big difference between the two problems. What if the actual entropy is very small so  $H'$  is also extremely small? In this case, the factor  $\frac{1}{H'}$  will be very large making the query procedure slow.

We overcome this technical difficulty by considering two cases. If  $H'$  is large, say  $H' \geq 0.9$ , then we can compute a multiplicative approximation of the entropy efficiently applying [23]. On the other hand, if  $H'$  is small, say  $H' < 0.9$ , then we use the ideas from [14] to design an efficient data structure. In particular, we check if there exists a value  $a_M$  with  $D(a_M) > 2/3$ .



If it does not exist then  $H'$  is large so it is easy to handle. If  $a_M$  exists, we write  $H(D)$  as a function of  $H(D \setminus \{a_M\})$  using Equation 4. In the end, if we get an additive approximation of  $H(D \setminus \{a_M\})$  we argue that this is sufficient to get a multiplicative approximation of  $H'$ .

**Data Structure.** For each color  $c_i$  we construct a range tree  $\mathcal{T}_i$  over  $P(u_i)$  as in the previous Subsection. Similarly, we construct a range tree  $\mathcal{T}$  over  $P$  for counting queries. We also construct the range tree  $\mathcal{S}$  for returning uniforms samples in a query rectangle. In addition to  $\mathcal{S}$ , we also construct a variation of this range tree, denoted by  $\bar{\mathcal{S}}$ . Given a query rectangle  $R$  and a color  $c_i$ ,  $\bar{\mathcal{S}}$  returns a point from  $\{p \in R \cap P \mid u(p) \neq c_i\}$  uniformly at random. In other words,  $\bar{\mathcal{S}}$  is a data structure over  $P$  that is used to return a point in a query rectangle uniformly at random excluding points of color  $c_i$ . While  $\bar{\mathcal{S}}$  is an extension of  $\mathcal{S}$ , the low level details are more tedious. We describe  $\bar{\mathcal{S}}$  in the full version of the paper [27].

The complexity of the proposed data structure is dominated by the complexity of  $\bar{\mathcal{S}}$ . Overall, it can be computed in  $O(n \log^d n)$  time and it has  $O(n \log^d n)$  space.

**Query procedure.** First, using  $\mathcal{T}$  we get  $N = |P \cap R|$ . Using  $\mathcal{S}$  we get  $\frac{\log(2n)}{\log 3}$  independent random samples from  $P \cap R$ . Let  $P_S$  be the set of returned samples. For each  $p \in P_S$  with  $u(p) = u_i$ , we run a counting query in  $\mathcal{T}_i$  to get  $N_i = |P(u_i) \cap R|$ . Finally, we check whether  $\frac{N_i}{N} > 2/3$ . If we do not find a point  $p \in P_S$  (assuming  $u(p) = u_i$ ) with  $\frac{N_i}{N} > 2/3$  then we run the algorithm from [23]. In particular, we set  $H' = 0.9$  and we run  $O\left(\frac{\log n}{\varepsilon^2 \cdot H'}\right)$  oracle queries  $\text{SAMP}_D$  or  $\text{EVAL}_D$ , as described in [23]. In the end we return the estimate  $h$ . Next, we assume that the algorithm found a point with color  $u_i$  satisfying  $\frac{N_i}{N} > 2/3$ . Using  $\bar{\mathcal{S}}$  (instead of  $\mathcal{S}$ ) we run the query procedure of the previous Subsection and we get an  $\varepsilon$ -additive approximation of  $H((P \setminus P(u_i)) \cap R)$ , i.e., the entropy of the points in  $P \cap R$  excluding points of color  $c_i$ . Let  $h'$  be the  $\varepsilon$ -additive approximation we get. In the end, we return the estimate  $h = \frac{N-N_i}{N} \cdot h' + \frac{N_i}{N} \log \frac{N}{N_i} + \frac{N-N_i}{N} \log \frac{N}{N-N_i}$ .

**Correctness.** It is straightforward to see that if there exists a color  $u_i$  containing more than  $2/3$ 's of all points in  $P \cap R$  then  $u_i \in u(P_S)$  with high probability. For completeness, in [27] we prove that this is the case with probability at least  $1 - 1/(2n)$ . Hence, with high probability, we make the correct decision.

If there is not such color, then in the full version [27] we show that the entropy in this case should be  $H(P \cap R) > 0.9$ . Hence,  $O\left(\frac{\log n}{\varepsilon^2}\right)$  oracle queries are sufficient to derive an  $(1 + \varepsilon)$ -multiplicative approximation of the correct entropy.

The interesting case is when we find a color  $u_i$  such that  $\frac{N_i}{N} > 2/3$  and  $\frac{N_i}{N} < 1$  (if  $\frac{N_i}{N} = 1$  then  $H(P \cap R) = 0$ ). Using the results of the previous Subsection along with the new data structure  $\bar{\mathcal{S}}$ , we get  $h' \in [H((P \setminus P(u_i)) \cap R) - \varepsilon, H((P \setminus P(u_i)) \cap R) + \varepsilon]$  with probability at least  $1 - 1/(2n)$ . We finally show that the estimate  $h$  we return is a multiplicative approximation of  $H(P \cap R)$ . From Equation 4, we have  $H(P \cap R) = \frac{N-N_i}{N} H((P \setminus P(u_i)) \cap R) + \frac{N_i}{N} \log \frac{N}{N_i} + \frac{N-N_i}{N} \log \frac{N}{N-N_i}$ . Since  $h' \in [H((P \setminus P(u_i)) \cap R) - \varepsilon, H((P \setminus P(u_i)) \cap R) + \varepsilon]$ , we get  $h \in [H(P \cap R) - \varepsilon \frac{N-N_i}{N_i}, H(P \cap R) + \varepsilon \frac{N-N_i}{N_i}]$ . If we show that  $\frac{N-N_i}{N_i} \leq H(P \cap R)$  then the result follows. By the definition of entropy we observe that  $H(P \cap R) \geq \frac{N_i}{N} \log \frac{N}{N_i} + \frac{N-N_i}{N} \log \frac{N}{N-N_i}$ . In the full version of the paper [27] we show that  $\frac{N-N_i}{N_i} \leq \frac{N_i}{N} \log \frac{N}{N_i} + \frac{N-N_i}{N} \log \frac{N}{N-N_i}$ , if  $1 > \frac{N_i}{N} > 2/3$ . We conclude that  $h \in [(1 - \varepsilon)H(P \cap R), (1 + \varepsilon)H(P \cap R)]$ .

**Analysis.** We first run a counting query on  $\mathcal{T}$  in  $O(\log^d n)$  time. Then the set  $P_S$  is constructed in  $O(\log^{d+1} n)$  time, running  $O(\log n)$  queries in  $\bar{\mathcal{S}}$ . In the first case of the query procedure (no point  $p$  with  $\frac{N_i}{N} > 2/3$ ) we run  $O\left(\frac{\log n}{\varepsilon^2}\right)$  oracle queries so in total it runs in

$O(\frac{\log^{d+1}}{\varepsilon^2})$  time. In the second case of the query procedure (point  $p$  with  $\frac{N_i}{N} > 2/3$ ) we run the query procedure of the previous Subsection using  $\bar{\mathcal{S}}$  instead of  $\mathcal{S}$ , so it takes  $O(\frac{\log^{d+3}}{\varepsilon^2})$  time. Overall, the query procedure takes  $O(\frac{\log^{d+3}}{\varepsilon^2})$  time.

► **Theorem 10.** *Let  $P$  be a set of  $n$  points in  $\mathbb{R}^d$ , where each point is associated with a color. A data structure of  $O(n \log^d n)$  size can be computed in  $O(n \log^d n)$  time, such that given a query hyper-rectangle  $R$  and a parameter  $\varepsilon \in (0, 1)$ , a value  $h$  can be computed in  $O(\frac{\log^{d+3} n}{\varepsilon^2})$  time, such that  $\frac{1}{1+\varepsilon}H(P \cap R) \leq h \leq (1+\varepsilon)H(P \cap R)$ , with high probability.*

This structure can be made dynamic under arbitrary insertions and deletions of points using well known techniques [9, 21, 33, 34]. The update time is  $O(\log^d n)$ .

### 5.3 Efficient additive and multiplicative approximation for $d = 1$

Next, for  $d = 1$ , we propose a deterministic, faster approximate data structure with query time  $O(\text{polylog } n)$  that returns an additive and multiplicative approximation of the entropy  $H(P \cap R)$ , given a query rectangle  $R$ .

Instead of using the machinery for entropy estimation on unknown distributions, we get the intuition from data structures that count the number of colors in a query region  $R$ . In [24], the authors presented a data structure to count/report colors in a query interval for  $d = 1$ . In particular, they map the range color counting/reporting problem for  $d = 1$  to the standard range counting/reporting problem in  $\mathbb{R}^2$ . Let  $P$  be the set of  $n$  colored points in  $\mathbb{R}^1$ . Let  $\bar{P} = \emptyset$  be the corresponding points in  $\mathbb{R}^2$  they construct. For every color  $u_i \in U$ , without loss of generality, let  $P(u_i) = \{p_1, p_2, \dots, p_k\}$  such that if  $j < \ell$  then the  $x$ -coordinate of point  $p_j$  is smaller than the  $x$ -coordinate of point  $p_\ell$ . For each point  $p_j \in P(u_i)$ , they construct the 2-d point  $\bar{p}_j = (p_j, p_{j-1})$  and they add it in  $\bar{P}$ . If  $p_j = p_1$ , then  $\bar{p}_1 = (p_1, -\infty)$ . Given a query interval  $R = [l, r]$  in 1-d, they map it to the query rectangle  $\bar{R} = [l, r] \times (-\infty, l)$ . It is straightforward to see that a point of color  $u_i$  exists in  $R$  if and only if  $\bar{R}$  contains exactly one transformed point of color  $u_i$ . Hence, using a range tree  $\bar{\mathcal{T}}$  on  $\bar{P}$  they can count (or report) the number of colors in  $P \cap R$  efficiently. While this is more than enough to count or report the colors in  $P \cap R$ , for the entropy we also need to know (in fact precompute) the number of points of each color  $u_i$  in  $P'$ , along with the actual entropy in each canonical subset. Notice that a canonical subset/node in  $\bar{\mathcal{T}}$  might belong to many different query rectangles  $\bar{R}$  that correspond to different query intervals  $R$ . Even though a point of color  $u_i$  appears only once in  $\bar{R} \cap \bar{P}$ , there can be multiple points with color  $u_i$  in  $R \cap P$ . Hence, there is no way to know in the preprocessing phase the exact number of points of each color presented in a canonical node of  $\bar{\mathcal{T}}$ . We overcome this technical difficulty by pre-computing for each canonical node  $v$  in  $\bar{\mathcal{T}}$ , monotone pairs with approximate values of (interval, number of points), and (interval, entropy) over a sufficiently large number of intervals. Another issue is that entropy is not monotone, so we split it into two monotone functions and we handle each of them separately until we merge them in the end to get the final estimation.

Before we start describing the data structure we prove some useful properties that we need later. For a set of colored points  $P' \subseteq P$ , with  $N = |P'|$ , let  $F(P') = N \cdot H(P') = \sum_{u_i \in u(P')} N_i \cdot \log \frac{N}{N_i}$ , where  $N_i$  is the number of points in  $P'$  with color  $u_i$ . We prove the next lemma in the full version [27].

► **Lemma 11.** *The function  $F(\cdot)$  is monotonically increasing. Furthermore,  $F(P') = O(N \log N)$ , and the smallest non-zero value that  $F(\cdot)$  can take is at least  $\log N$ .*



**Data structure.** We apply the same mapping from  $P$  to  $\bar{P}$  as described above [24] and construct a range tree  $\bar{\mathcal{T}}$  on  $\bar{P}$ . Then we visit each canonical node  $v$  of  $\bar{\mathcal{T}}$ . If node  $v$  contains two points with the same color then we can skip it because this node will not be returned as a canonical node for any query  $\bar{R}$ . Let  $v$  be a node such that  $\bar{P}_v$  does not contain two points with the same color. Let also  $x_v$  be the smallest  $x$ -coordinate of a point in  $\bar{P}_v$ . Finally, let  $U_v = u(\bar{P}_v)$ , and  $P(U_v) = \{p \in P \mid u(p) \in U_v\}$ . Notice that  $P(U_v)$  is a subset of  $P$  and not of  $\bar{P}$ . We initialize an empty array  $S_v$  of size  $O(\frac{\log n}{\varepsilon})$ . Each element  $S_v[i]$  stores the maximum  $x$  coordinate such that  $(1 + \varepsilon)^i \geq |P(U_v) \cap [x_v, x]|$ . Furthermore, we initialize an empty array  $H_v$  of size  $O(\frac{\log n}{\varepsilon})$ . Each element  $H_v[i]$  stores the maximum  $x$  coordinate such that  $(1 + \varepsilon)^i \geq F(P(U_v) \cap [x_v, x])$ . We notice that both functions  $F(\cdot)$ , and cardinality of points are monotonically increasing. For every node of  $\bar{\mathcal{T}}$  we use  $O(\frac{\log n}{\varepsilon})$  space, so in total, the space of our data structure is  $O(\frac{n}{\varepsilon} \log^2 n)$ . In the full version of the paper [27] we show how we can construct the data structure  $\bar{\mathcal{T}}$  in  $O(\frac{n}{\varepsilon} \log^5 n)$  time.

**Query procedure.** Given a query interval  $R = [a, b]$ , we run a query in  $\bar{\mathcal{T}}$  using the query range  $\bar{R}$ . Let  $V = \{v_1, \dots, v_k\}$  be the set of  $k = O(\log^2 n)$  returned canonical nodes. For each node  $v \in V$  we run a binary search in array  $S_v$  and a binary search in  $H_v$  with key  $b$ . Let  $\ell_v^S$  be the minimum index such that  $b \leq S_v[\ell_v^S]$  and  $\ell_v^H$  be the minimum index such that  $b \leq H_v[\ell_v^H]$ . From their definitions, it holds that  $|P(U_v) \cap R| \leq (1 + \varepsilon)^{\ell_v^S} \leq (1 + \varepsilon)|P(U_v) \cap R|$ , and  $F(P(U_v) \cap R) \leq (1 + \varepsilon)^{\ell_v^H} \leq (1 + \varepsilon)F(P(U_v) \cap R)$ . Hence, we can approximate the entropy of  $P(U_v) \cap R$ , defining  $\mathcal{H}_v = \frac{(1 + \varepsilon)^{\ell_v^H}}{(1 + \varepsilon)^{\ell_v^S - 1}}$ . The next Lemma shows that  $\mathcal{H}_v$  is a good approximation of  $H(P(U_v) \cap R)$ .

► **Lemma 12.** *It holds that  $H(P(U_v) \cap R) \leq \mathcal{H}_v \leq (1 + \varepsilon)^2 H(P(U_v) \cap R)$ .*

**Proof.** We have  $\mathcal{H}_v = \frac{(1 + \varepsilon)^{\ell_v^H}}{(1 + \varepsilon)^{\ell_v^S - 1}}$ . From their definitions, we have that  $|P(U_v) \cap R| \leq (1 + \varepsilon)^{\ell_v^S} \leq (1 + \varepsilon)|P(U_v) \cap R|$ , and  $F(P(U_v) \cap R) \leq (1 + \varepsilon)^{\ell_v^H} \leq (1 + \varepsilon)F(P(U_v) \cap R)$ . It also holds that  $(1 + \varepsilon)^{\ell_v^S - 1} \leq |P(U_v) \cap R|$  and  $(1 + \varepsilon)^{\ell_v^S - 1} \geq \frac{|P(U_v) \cap R|}{(1 + \varepsilon)}$ . Hence  $\mathcal{H}_v \leq \frac{(1 + \varepsilon)^{\ell_v^H} F(P(U_v) \cap R)}{|P(U_v) \cap R| / (1 + \varepsilon)} \leq (1 + \varepsilon)^2 H(P(U_v) \cap R)$ . Furthermore,  $\mathcal{H}_v \geq \frac{F(P(U_v) \cap R)}{|P(U_v) \cap R|} = H(P(U_v) \cap R)$ . ◀

We find the overall entropy by merging together pairs of canonical nodes. Notice that we can do it easily using Equation 2 because all colors are different between any pair of nodes in  $V$ . For example, we apply Equation 2 for two nodes  $v, w \in V$  as follows:

$$\frac{(1 + \varepsilon)^{\ell_v^S} \mathcal{H}_v + (1 + \varepsilon)^{\ell_w^S} \mathcal{H}_w + (1 + \varepsilon)^{\ell_v^S} \log \left( \frac{(1 + \varepsilon)^{\ell_v^S} + (1 + \varepsilon)^{\ell_w^S}}{(1 + \varepsilon)^{\ell_v^S - 1}} \right) + (1 + \varepsilon)^{\ell_w^S} \log \left( \frac{(1 + \varepsilon)^{\ell_v^S} + (1 + \varepsilon)^{\ell_w^S}}{(1 + \varepsilon)^{\ell_w^S - 1}} \right)}{(1 + \varepsilon)^{\ell_v^S - 1} + (1 + \varepsilon)^{\ell_w^S - 1}}.$$

In the end we compute the overall entropy  $\mathcal{H}$ . The next Lemma shows the correctness of our procedure.

► **Lemma 13.** *If we set  $\varepsilon \leftarrow \frac{\varepsilon}{4 \cdot c \cdot \log n}$ , it holds that  $H(P \cap R) \leq \mathcal{H} \leq (1 + \varepsilon)H(P \cap R) + \varepsilon$ , for a constant  $c > 0$ .*

**Proof.** We assume that we take the union of two nodes  $v, w \in V$  using Equation 2. We can use this equation because nodes  $v, w$  do not contain points with similar colors. Let  $H_1 = H(P(U_v) \cap R)$ ,  $H_2 = H(P(U_w) \cap R)$ ,  $N_1 = |P(U_v) \cap R|$ , and  $N_2 = |P(U_w) \cap R|$ . We have

$$\mathcal{H}_{v,w} = \frac{(1 + \varepsilon)^{\ell_v^S} \mathcal{H}_v + (1 + \varepsilon)^{\ell_w^S} \mathcal{H}_w + (1 + \varepsilon)^{\ell_v^S} \log \left( \frac{(1 + \varepsilon)^{\ell_v^S} + (1 + \varepsilon)^{\ell_w^S}}{(1 + \varepsilon)^{\ell_v^S - 1}} \right) + (1 + \varepsilon)^{\ell_w^S} \log \left( \frac{(1 + \varepsilon)^{\ell_v^S} + (1 + \varepsilon)^{\ell_w^S}}{(1 + \varepsilon)^{\ell_w^S - 1}} \right)}{(1 + \varepsilon)^{\ell_v^S - 1} + (1 + \varepsilon)^{\ell_w^S - 1}}.$$

## 6:16 Range Entropy Queries and Partitioning

Using Lemma 12, we get

$$\mathcal{H}_{v,w} \leq \frac{(1+\varepsilon)^4 N_1 H_1 + (1+\varepsilon)^4 N_2 H_2 + (1+\varepsilon)^2 N_1 \log\left((1+\varepsilon)^2 \frac{N_1+N_2}{N_1}\right) + (1+\varepsilon)^2 N_2 \log\left((1+\varepsilon)^2 \frac{N_1+N_2}{N_2}\right)}{N_1+N_2}$$

and we conclude that

$$\mathcal{H}_{v,w} \leq (1+\varepsilon)^4 H((P(U_v) \cup P(U_w)) \cap R) + (1+\varepsilon)^2 \log(1+\varepsilon)^2.$$

Similarly if we have computed  $\mathcal{H}_{x,y}$  for two other nodes  $x, y \in V$ , then

$$\mathcal{H}_{x,y} \leq (1+\varepsilon)^4 H((P(U_x) \cup P(U_y)) \cap R) + (1+\varepsilon)^2 \log(1+\varepsilon)^2.$$

If we compute their union, we get

$$\mathcal{H}_{v,w,x,y} \leq (1+\varepsilon)^6 H((P(U_v) \cup P(U_w) \cup P(U_x) \cup P(U_y)) \cap R) + [(1+\varepsilon)^4 + (1+\varepsilon)^2] \log(1+\varepsilon)^2.$$

In the end of this process we have

$$\mathcal{H} \geq H(P \cap R)$$

because all intermediate estimations of entropy are larger than the actual entropy. For a constant  $c$ , it also holds that

$$\mathcal{H} \leq (1+\varepsilon)^{c \log(\log n)} H(P \cap R) + \sum_{j=1}^{c \log(\log n)/2} (1+\varepsilon)^{2j} \log(1+\varepsilon)^2.$$

This quantity can be bounded by

$$\mathcal{H} \leq (1+\varepsilon)^{c \log(\log n)} H(P \cap R) + c \log(\log n) (1+\varepsilon)^{c \log(\log n)} \log(1+\varepsilon).$$

We have the factor  $\log(\log n)$  because  $|V| = O(\log^2 n)$  so the number of levels of recurrence is  $O(\log(\log n))$ .

Next, we show that if we set  $\varepsilon \leftarrow \frac{\varepsilon}{4 \cdot c \log(\log n)}$ , then  $\mathcal{H} \leq (1+\varepsilon)H(P \cap R) + \varepsilon$ .

We have

$$\left(1 + \frac{\varepsilon/4}{c \log(\log n)}\right)^{c \log(\log n)} \leq e^{\varepsilon/4} \leq 1 + \varepsilon.$$

The first inequality holds because of the well known inequality  $(1+x/n)^n \leq e^x$ . The second inequality is always true for  $\varepsilon \in (0, 1)$ . Then we have

$$(1+\varepsilon)c \log(\log n) \log\left(1 + \frac{\varepsilon}{4 \cdot c \log(\log n)}\right) \leq 2c \log(\log n) \log\left(1 + \frac{\varepsilon}{4 \cdot c \log(\log n)}\right).$$

Next, we show that this quantity is at most  $\varepsilon$ . Let  $L = c \log(\log n)$  and let

$$f(x) = x - 2L \log\left(1 + \frac{x}{4L}\right)$$

be a real function for  $x \in [0, 1]$ . We have

$$f'(x) = 1 - \frac{2L}{L \ln(16) + x \ln(2)}.$$

We observe that  $\ln(16) \approx 2.77$  and  $x \ln(2) \geq 0$  so  $f'(x) \geq 0$  and  $f$  is monotonically increasing. So  $f(x) \geq f(0) = 0$ . Hence, for any  $\varepsilon \in [0, 1]$  we have

$$\varepsilon - 2L \log\left(1 + \frac{\varepsilon}{4L}\right) \geq 0.$$

We conclude with

$$\mathcal{H} \leq (1+\varepsilon)H(P \cap R) + \varepsilon. \quad \blacktriangleleft$$

We need  $O(\log^2 n)$  time to get  $V$  from  $\bar{\mathcal{T}}$ . Then, we run binary search for each node  $v \in V$  so we spend  $O(\log^2 n \log \frac{\log n \log \log n}{\varepsilon}) = O(\log^2 n \log \frac{\log n}{\varepsilon})$  time. We merge and update the overall entropy in time  $O(|V|)$ , so in total the query time is  $O(\log^2 n \log \frac{\log n}{\varepsilon})$ .

► **Theorem 14.** *Let  $P$  be a set of  $n$  points in  $\mathbb{R}^1$ , where each point is associated with a color, and let  $\varepsilon \in (0, 1)$  be a parameter. A data structure of  $O(\frac{n}{\varepsilon} \log^2 n)$  size can be computed in  $O(\frac{n}{\varepsilon} \log^5 n)$  time, such that given a query hyper-rectangle  $R$ , a value  $h$  can be computed in  $O(\log^2 n \log \frac{\log n}{\varepsilon})$  time, such that  $H(P \cap R) \leq h \leq (1 + \varepsilon)H(P \cap R) + \varepsilon$ .*

## 6 Partitioning

The new data structures can be used to accelerate some partitioning algorithms with respect to the (expected) entropy. Let DS be one of our new data structures over  $n$  items that can be constructed in  $O(P(n))$  time, has  $O(S(n))$  space, and given a query range  $R$ , returns a value  $h$  in  $O(Q(n))$  time such that  $\frac{1}{\alpha}H - \beta \leq h \leq \alpha \cdot H + \beta$ , where  $H$  is the entropy of the items in  $R$ , and  $\alpha \geq 1$ ,  $\beta \geq 0$  two error thresholds. On the other hand, the straightforward way to compute the (expected) entropy without using any data structure has preprocessing time  $O(1)$ , query time  $O(n)$  and it returns the exact entropy in a query range.

In most cases we consider the expected entropy to partition the dataset as this is mostly the case in entropy-based partitioning and clustering algorithms. Except of being a useful quantity bounding both the uncertainty and the size of a bucket, it is also monotone. All our data structures can work for both the entropy and expected entropy quantity almost verbatim. We define two optimization problems. Let **MaxPart** be the problem of constructing a partitioning with  $k$  buckets that maximizes/minimizes the maximum (expected) entropy in a bucket. Let **SumPart** be the problem of constructing a partitioning with  $k$  buckets that maximizes/minimizes the sum of (expected) entropies over the buckets. For simplicity, in order to compare the running times, we skip the  $\log(n)$  factors from the running times.

**Partitioning for  $d = 1$ .** We can easily solve **MaxPart** using dynamic programming:  $DP[i, j] = \min_{\ell < i} \max\{DP[i - \ell, j - 1], \text{Error}[i - \ell + 1, i]\}$ , where  $DP[i, j]$  is the minimum max entropy of the first  $i$  items using  $j$  buckets, and  $\text{Error}[i, j]$  is the expected entropy among the items  $i$  and  $j$ . Since  $\text{Error}$  is monotone, we can find the optimum  $DP[i, j]$  running a binary search on  $\ell$ , i.e., we do not need to visit all indexes  $\ell < i$  one by one to find the optimum. Without using any data structure the running time to find  $DP[n, k]$  is  $O(kn^2)$ . Using DS, the running time for partitioning is  $O(P(n) + knQ(n))$ . If we use the data structure from Section 4.1 for  $t = 0.5$ , then the running time is  $O(kn\sqrt{n}) = o(kn^2)$ .

Next we consider approximation algorithms for the **MaxPart** and **SumPart** problems.

It is easy to observe that the maximum value and the minimum non-zero value of the optimum solution of **MaxPart** are bounded polynomially on  $n$ . Let  $[l_M, r_M]$  be the range of the optimum values. We discretize the range  $[l_M, r_M]$  by a multiplicative factor  $(1 + \varepsilon)$ . We run a binary search on the discrete values. For each value  $e \in [l_M, r_M]$  we consider, we construct a new bucket by running another binary search on the input items, trying to expand the bucket until its expected entropy is at most  $e$ . We repeat the same for all buckets and we decide if we should increase or decrease the error  $e$  in the next iteration. In the end the solution we find is within an  $(1 + \varepsilon)$  factor far from the max expected entropy in the optimum partitioning. Without using any data structure, we need  $O(n \log \frac{1}{\varepsilon})$  time to construct the partitioning. If we use DS we need time  $O(P(n) + kQ(n) \log \frac{1}{\varepsilon})$ . If we use the data structure

in Subsection 5.2 we have partition time  $O\left(n + \frac{k}{\varepsilon^2} \log \frac{1}{\varepsilon}\right) = o\left(n \log \frac{1}{\varepsilon}\right)$ . If we allow a  $\Delta$  additive approximation in addition to the  $(1 + \varepsilon)$  multiplicative approximation, we can use the data structure in Subsection 5.1 having partition time  $O\left(n + \frac{k}{\Delta^2} \log \frac{1}{\varepsilon}\right) = o\left(n \log \frac{1}{\varepsilon}\right)$ .

Next, we focus on the SumPart problem. It is known from [22] (Theorems 5, 6) that if the error function is monotone (such as the expected entropy) then we can get a partitioning with  $(1 + \varepsilon)$ -multiplicative approximation in  $O\left(P(n) + \frac{k^3}{\varepsilon^2} Q(n)\right)$  time. Hence, the straightforward solution without using a data structure returns an  $(1 + \varepsilon)$ -approximation of the optimum partitioning in  $O\left(\frac{k^3}{\varepsilon^2} n\right)$  time. If we use the data structure from Subsection 5.2 we have running time  $O\left(n + \frac{k^3}{\varepsilon^4}\right)$ , which is  $o\left(\frac{k^3}{\varepsilon^2} n\right)$  and multiplicative error  $(1 + \varepsilon)^2$ . If we set  $\varepsilon \leftarrow \varepsilon/3$  then in the same asymptotic running time we have error  $(1 + \varepsilon)$ . If we also allow  $\Delta \cdot n$  additive approximation, we can use the additive approximation DS from Subsection 5.1. The running time will be  $O\left(n + \frac{k^3}{\varepsilon^2 \Delta^2}\right) = o\left(\frac{k^3}{\varepsilon^2} n\right)$ .

**Partitioning for  $d > 1$ .** Partitioning and constructing histograms in high dimensions is usually a very challenging task, since most of the known algorithms with theoretical guarantees are very expensive [18]. However, there is a practical method with some conditional error guarantees, that works very well in any constant dimension  $d$  and it has been used in a few papers [5, 30, 31]. The idea is to construct a tree having a rectangle containing all points in the root. In each iteration of the algorithm, we choose to split (on the median in each coordinate or find the best split) the (leaf) node with the minimum/maximum (expected) entropy. As stated in previous papers, let make the assumption that an optimum algorithm for either MaxPart or SumPart is an algorithm that always chooses to split the leaf node with the smallest/largest expected entropy. Using the straightforward solution without data structures, we can construct an “optimum” partitioning in  $O(kn)$  time by visiting all points in every new generated rectangle. Using DS, the running time of the algorithm is  $O(P(n) + kQ(n))$ . In order to get an optimum solution we use DS from Subsection 4.2. The overall running time is  $O(n^{(2d-1)t+1} + kn^{1-t})$ . This is minimized for  $n^{(2d-1)t+1} = kn^{1-t} \Leftrightarrow t = t^* = \frac{\log k}{2d \log n}$ , so the overall running time is  $O(kn^{1-t^*}) = o(kn)$ . If we allow  $(1 + \varepsilon)$ -multiplicative approximation we can use the DS from Subsection 5.2. The running time will be  $O\left(n + \frac{k}{\varepsilon^2}\right) = o(kn)$ . If we allow a  $\Delta$ -additive approximation, then we can use the DS from Subsection 5.1 with running time  $O\left(n + \frac{k}{\Delta^2}\right) = o(kn)$ .

## 7 Conclusion

In this work, we presented efficient data structures for computing (exactly and approximately) the entropy of the points in a rectangular query in sub-linear time. Using our new data structures we can accelerate partitioning algorithms for columnar compression (Example 1) and histogram construction (Example 2). Furthermore, we can accelerate the exploration of high uncertainty regions for data cleaning (Example 3).

There are multiple interesting open problems derived from this work. i) Our approximate data structures are dynamic but our exact data structures are static. Is it possible to have dynamic data structure for returning the exact entropy? ii) We showed a lower bound for designing exact data structures when  $P \in \mathbb{R}^d$  for  $d \geq 2$ . Does the lower bound extend for  $d = 1$ ? iii) There is still a gap between the proposed lower bound and upper bound. An interesting problem is to close that gap. iv) Can we extend the faster deterministic approximation data structure from Subsection 5.3 in higher dimensions?

## References

- 1 Peyman Afshani and Jeff M Phillips. Independent range sampling, revisited again. In *35th International Symposium on Computational Geometry (SoCG 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019. doi:10.4230/LIPIcs.SoCG.2019.4.
- 2 Peyman Afshani and Zhewei Wei. Independent range sampling, revisited. In *25th Annual European Symposium on Algorithms (ESA 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.ESA.2017.3.
- 3 Pankaj K Agarwal, Nirman Kumar, Stavros Sintos, and Subhash Suri. Range-max queries on uncertain data. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 465–476, 2016. doi:10.1145/2902251.2902281.
- 4 Pankaj K Agarwal, Nirman Kumar, Stavros Sintos, and Subhash Suri. Range-max queries on uncertain data. *Journal of Computer and System Sciences*, 94:118–134, 2018. doi:10.1016/j.jcss.2017.09.006.
- 5 Linas Baltrunas, Arturas Mazeika, and Michael Bohlen. Multi-dimensional histograms with tight bounds for the error. In *2006 10th International Database Engineering and Applications Symposium (IDEAS'06)*, pages 105–112. IEEE, 2006. doi:10.1109/IDEAS.2006.31.
- 6 Daniel Barbará, Yi Li, and Julia Couto. Coolcat: an entropy-based algorithm for categorical clustering. In *Proceedings of the eleventh international conference on Information and knowledge management*, pages 582–589, 2002. doi:10.1145/584792.584888.
- 7 Tuğkan Batu, Sanjoy Dasgupta, Ravi Kumar, and Ronitt Rubinfeld. The complexity of approximating entropy. In *Proceedings of the thirty-fourth annual ACM symposium on Theory of computing*, pages 678–687, 2002. doi:10.1145/509907.510005.
- 8 Irad Ben-Gal, Shahar Weinstock, Gonen Singer, and Nicholas Bambos. Clustering users by their mobility behavioral patterns. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 13(4):1–28, 2019. doi:10.1145/3322126.
- 9 Jon Louis Bentley and James B Saxe. Decomposable searching problems i. static-to-dynamic transformation. *Journal of Algorithms*, 1(4):301–358, 1980. doi:10.1016/0196-6774(80)90015-2.
- 10 Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. Computational geometry. In *Computational geometry*, pages 1–17. Springer, 1997.
- 11 Lakshminath Bhuvanagiri and Sumit Ganguly. Estimating entropy over data streams. In *Algorithms-ESA 2006: 14th Annual European Symposium, Zurich, Switzerland, September 11-13, 2006. Proceedings 14*, pages 148–159. Springer, 2006. doi:10.1007/11841036\_16.
- 12 Cafer Caferov, Barış Kaya, Ryan O'Donnell, and AC Say. Optimal bounds for estimating entropy with pmf queries. In *International Symposium on Mathematical Foundations of Computer Science*, pages 187–198. Springer, 2015. doi:10.1007/978-3-662-48054-0\_16.
- 13 Clément Canonne and Ronitt Rubinfeld. Testing probability distributions underlying aggregated data. In *International Colloquium on Automata, Languages, and Programming*, pages 283–295. Springer, 2014. doi:10.1007/978-3-662-43948-7\_24.
- 14 Amit Chakrabarti, Graham Cormode, and Andrew McGregor. A near-optimal algorithm for computing the entropy of a stream. In *SODA*, volume 7, pages 328–335. Citeseer, 2007. URL: <http://dl.acm.org/citation.cfm?id=1283383.1283418>.
- 15 Amit Chakrabarti, Khanh Do Ba, and S Muthukrishnan. Estimating entropy and entropy norm on data streams. *Internet Mathematics*, 3(1):63–78, 2006. doi:10.1080/15427951.2006.10129117.
- 16 Xu Chu, John Morcos, Ihab F Ilyas, Mourad Ouzzani, Paolo Papotti, Nan Tang, and Yin Ye. Katara: A data cleaning system powered by knowledge bases and crowdsourcing. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, pages 1247–1261, 2015. doi:10.1145/2723372.2749431.
- 17 Peter Clifford and Ioana Cosma. A simple sketching algorithm for entropy estimation over streaming data. In *Artificial Intelligence and Statistics*, pages 196–206. PMLR, 2013. URL: <http://proceedings.mlr.press/v31/clifford13a.html>.

- 18 Graham Cormode, Minos Garofalakis, Peter J Haas, Chris Jermaine, et al. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends® in Databases*, 4(1–3):1–294, 2011. doi:10.1561/19000000004.
- 19 Juan David Cruz, Cécile Bothorel, and François Poulet. Entropy based community detection in augmented social networks. In *2011 International Conference on computational aspects of social networks (CASoN)*, pages 163–168. IEEE, 2011. doi:10.1109/CASON.2011.6085937.
- 20 Pooya Davoodi, Michiel Smid, and Freek van Walderveen. Two-dimensional range diameter queries. In *Latin American Symposium on Theoretical Informatics*, pages 219–230. Springer, 2012. doi:10.1007/978-3-642-29344-3\_19.
- 21 J. Erickson. Static-to-dynamic transformations. <http://jeffe.cs.illinois.edu/teaching/datastructures/notes/01-statictodynamic.pdf>.
- 22 Sudipto Guha, Nick Koudas, and Kyuseok Shim. Approximation and streaming algorithms for histogram construction problems. *ACM Transactions on Database Systems (TODS)*, 31(1):396–438, 2006. doi:10.1145/1132863.1132873.
- 23 Sudipto Guha, Andrew McGregor, and Suresh Venkatasubramanian. Streaming and sublinear approximation of entropy and information distances. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 733–742, 2006. URL: <http://dl.acm.org/citation.cfm?id=1109557.1109637>.
- 24 Prosenjit Gupta, Ravi Janardan, and Michiel Smid. Further results on generalized intersection searching problems: counting, reporting, and dynamization. *Journal of Algorithms*, 19(2):282–317, 1995. doi:10.1006/jagm.1995.1038.
- 25 Nicholas JA Harvey, Jelani Nelson, and Krzysztof Onak. Sketching and streaming entropy via approximation theory. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 489–498. IEEE, 2008. doi:10.1109/FOCS.2008.76.
- 26 Xiaocheng Hu, Miao Qiao, and Yufei Tao. Independent range sampling. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 246–255, 2014. doi:10.1145/2594538.2594545.
- 27 Sanjay Krishnan and Stavros Sintos. Range entropy queries and partitioning. *CoRR*, abs/2312.15959, 2023. doi:10.48550/arXiv.2312.15959.
- 28 Ping Li and Cun-Hui Zhang. A new algorithm for compressed counting with applications in shannon entropy estimation in dynamic data. In *Proceedings of the 24th Annual Conference on Learning Theory*, pages 477–496. JMLR Workshop and Conference Proceedings, 2011. URL: <http://proceedings.mlr.press/v19/li11a/li11a.pdf>.
- 29 Tao Li, Sheng Ma, and Mitsunori Ogihara. Entropy-based criterion in categorical clustering. In *Proceedings of the twenty-first international conference on Machine learning*, page 68, 2004. doi:10.1145/1015330.1015404.
- 30 Xi Liang, Stavros Sintos, and Sanjay Krishnan. JanusAQP: Efficient partition tree maintenance for dynamic approximate query processing. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*, pages 572–584. IEEE, 2023. doi:10.1109/ICDE55515.2023.00050.
- 31 Xi Liang, Stavros Sintos, Zechao Shang, and Sanjay Krishnan. Combining aggregation and sampling (nearly) optimally for approximate query processing. In *Proceedings of the 2021 International Conference on Management of Data*, pages 1129–1141, 2021. doi:10.1145/3448016.3457277.
- 32 Andres Lopez Martinez. Parallel minimum cuts: An improved crew pram algorithm. *Master's thesis. KTH, School of Electrical Engineering and Computer Science (EECS)*, 2020.
- 33 Mark H Overmars. *The design of dynamic data structures*, volume 156. Springer Science & Business Media, 1983. doi:10.1007/BFb0014927.
- 34 Mark H Overmars and Jan van Leeuwen. Worst-case optimal insertion and deletion methods for decomposable searching problems. *Information Processing Letters*, 12(4):168–173, 1981. doi:10.1016/0020-0190(81)90093-4.

- 35 Mihai Patrascu and Liam Roditty. Distance oracles beyond the thorup-zwick bound. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 815–823. IEEE, 2010. doi:10.1109/FOCS.2010.83.
- 36 Saladi Rahul and Ravi Janardan. Algorithms for range-skyline queries. In *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*, pages 526–529, 2012. doi:10.1145/2424321.2424406.
- 37 Yufei Tao. Algorithmic techniques for independent query sampling. In *Proceedings of the 41st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 129–138, 2022. doi:10.1145/3517804.3526068.
- 38 Hien To, Kuorong Chiang, and Cyrus Shahabi. Entropy-based histograms for selectivity estimation. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 1939–1948, 2013. doi:10.1145/2505515.2505756.
- 39 Lu Wang, Robert Christensen, Feifei Li, and Ke Yi. Spatial online sampling and aggregation. *Proceedings of the VLDB Endowment*, 9(3):84–95, 2015. doi:10.14778/2850583.2850584.
- 40 Dong Xie, Jeff M Phillips, Michael Matheny, and Feifei Li. Spatial independent range sampling. In *Proceedings of the 2021 International Conference on Management of Data*, pages 2023–2035, 2021. doi:10.1145/3448016.3452806.







# Skyline Operators for Document Spanners

Antoine Amarilli   

LTCI, Télécom Paris, Institut Polytechnique de Paris, France

Benny Kimelfeld  

Technion – Israel Institute of Technology, Haifa, Israel

Sébastien Labbé

École normale supérieure, Paris, France

Stefan Mengel

Univ. Artois, CNRS, Centre de Recherche en Informatique de Lens (CRIL), France

---

## Abstract

When extracting a relation of spans (intervals) from a text document, a common practice is to filter out tuples of the relation that are deemed dominated by others. The domination rule is defined as a partial order that varies along different systems and tasks. For example, we may state that a tuple is dominated by tuples that extend it by assigning additional attributes, or assigning larger intervals. The result of filtering the relation would then be the *skyline* according to this partial order. As this filtering may remove most of the extracted tuples, we study whether we can improve the performance of the extraction by compiling the domination rule into the extractor.

To this aim, we introduce the *skyline operator* for declarative information extraction tasks expressed as document spanners. We show that this operator can be expressed via regular operations when the domination partial order can itself be expressed as a regular spanner, which covers several natural domination rules. Yet, we show that the skyline operator incurs a computational cost (under combined complexity). First, there are cases where the operator requires an exponential blowup on the number of states needed to represent the spanner as a sequential variable-set automaton. Second, the evaluation may become computationally hard. Our analysis more precisely identifies classes of domination rules for which the combined complexity is tractable or intractable.

**2012 ACM Subject Classification** Information systems → Information extraction; Theory of computation → Database query processing and optimization (theory)

**Keywords and phrases** Information Extraction, Document Spanners, Query Evaluation

**Digital Object Identifier** 10.4230/LIPIcs.ICDT.2024.7

**Related Version** *Full Version (with all Proofs)*: <https://arxiv.org/abs/2304.06155> [4]

**Funding** The work of Amarilli, Labbé, and Mengel were partially supported by the ANR project EQUUS ANR-19-CE48-0019. The work of Kimelfeld was supported by the Israel Science Foundation (ISF) Grant 768/19.

## 1 Introduction

The framework of *document spanners* [10] is an established formalism to express declarative information extraction tasks. A *spanner* specifies the possible ways to assign variables over a textual document, producing so-called *mappings* which are the result of the extraction: each mapping assigns the variables to a factor of the document, called a *span*. The spanner formalism has been defined in terms of several operators, in particular regular operations extended with capture variables (corresponding to so-called *regular spanners*), operators from relational algebra (which can sometimes be translated into regular expressions), string equality (the so-called *core spanners*), etc.



© Antoine Amarilli, Benny Kimelfeld, Sébastien Labbé, and Stefan Mengel; licensed under Creative Commons License CC-BY 4.0

27th International Conference on Database Theory (ICDT 2024).

Editors: Graham Cormode and Michael Shekelyan; Article No. 7; pp. 7:1–7:18

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Tuples extracted from text often aim to capture mentions of real-life entities and relationships. In that respect, one of the studied challenges is that different extracted tuples might be considered as conflicting with each other [11]. A common example is that of overlapping spans; for instance, a situation where one entity mention is contained within another entity mention is considered inconsistent. For this reason, traditional declarative systems for information extraction provide explicit mechanisms for restricting the extracted spans to the *maximal* ones according to different comparisons. IBM’s SystemT [17] has the *consolidation* rules such as “contained-within” (where a span dominates its subspans) and “left-to-right” (where a span dominates all shorter spans that begin at the same position). Similarly, the GATE system [9] features *controls* such as “Appelt” (which is similar to SystemT’s “left-to-right”). Alternatively, in the *schemaless* context of document spanners where we can assign spans to only a subset of variables [18], we may want to only capture spans that assign a maximal subset of the variables and cannot be extended by assigning more variables; in the spirit, for instance, of the relational *full disjunction* [14] or the *OPTIONAL* operator of SPARQL [2].

To explore the expressive power of operators such as controls and consolidators, Fagin et al. [11] proposed a framework that enriches document spanners with a previous concept of *prioritized repairs* [26]. There, they defined the notion of a “denial preference-generating dependency” (denial pgd) that expresses the binary domination relationship using the underlying spanner language. When this relationship is transitive, the result of applying the denial pgd is precisely the set of maximal tuples. However, they did not address the computational complexity of this operator and, consequently, it has been left open. (Moreover, their study does not apply to the schemaless context.)

The notion of maximal matches has been abundantly studied in other areas of database research, where it is called the *skyline operator* [5]. Intuitively, the skyline of a set of results under a partial order relation is the set of the results that are maximal, i.e., are not dominated by another result. The complexity of skyline computation has been investigated under many dimensions, e.g., I/O access [25], parallel computation [1], or noisy comparisons [16]. However, we are not aware of a study of the complexity of this operator to extract the maximal matches of document spanners. This is the focus of the present paper.

**Contributions.** We present our contributions together with the structure of the paper. After some necessary preliminaries (Section 2), we first introduce in Section 3 the skyline operator. The operator is defined as extracting the maximal mappings of a spanner on a document with respect to a partial order on the mappings, which we call a *domination relation*. In particular, we define the *span inclusion*, *span length*, *variable inclusion*, and *left-to-right domination relations*, which cover the examples presented above.

To allow for a unified study of these operators, and similarly to [11], we propose a general model where the domination relations are themselves expressed as document spanners. More precisely, a *domination rule* is a spanner that defines a domination relation on every document: it indicates which mappings dominate which other mappings, by intuitively capturing pairs  $(m, m')$  that indicate that  $m'$  dominates  $m$ . We also focus on so-called *variable-wise rules*, where the domination relation on mappings can be defined as a product of relations on spans. In other words, a variable-wise rule is a spanner expressing which spans dominate which spans, and the domination relation on mappings is obtained in a pointwise fashion across the variables, like the *ceteris paribus* semantics for preference handling in artificial intelligence [6] or Pareto-optimal points for skyline queries on multidimensional data [16]. All examples introduced earlier can be expressed in this variable-wise way.

We then begin our study of how to evaluate the skyline operator on document spanners, and start in Section 4 with the question of *expressiveness*: does the operator strictly increase the expressive power of spanner formalisms, or can it be rewritten using existing operators? We show that *regular spanners* are closed under the skyline operator, generalizing a result of [11] to the schemaless context. By contrast, we show that *core spanners* are not closed under skylines, even for the fixed variable inclusion or span inclusion domination relations, again generalizing a result of [11].

Next, we explore the question of whether it is possible to tractably rewrite the skyline operator into regular spanners, to allow for efficient evaluation like, e.g., the polynomial-time compilation of the join operator in the schema-based context (see [19], Lemma 4.4.7). We present in Section 5 a lower bound establishing that this is not the case: even for variable inclusion domination, applying the skyline operator to a spanner expressed as a sequential variable-set automaton (VA) incurs a necessary exponential blowup. This result is shown by identifying a connection between VAs and *nondeterministic read-once branching programs* (NROBPs). This general-purpose method can be used outside of the context of skylines, and in fact we also use it to show a result of independent interest: there are regex-formulas on which the *natural join* operator incurs an unavoidable exponential blowup (Theorem 5.5).

We then move in Section 6 from state complexity to the *computational complexity* of skyline evaluation for regular spanners. This task is clearly tractable in *data complexity*, i.e., for a fixed spanner and domination rule: we simply compute all captured mappings, and filter out the non-maximal ones. More interestingly, assuming  $P \neq NP$ , we show that the task is intractable in *combined complexity*, i.e., as a function of the input spanner (Theorem 6.3), already in the case of the variable inclusion relation. Hence, we cannot tractably evaluate the skyline operator in combined complexity, even without compiling it to an explicit VA.

Lastly, we study in more detail how the complexity of skyline computation depends on the fixed domination relation: are there non-trivial domination rules for which skyline computation is tractable in combined complexity? We show in Section 7 a sufficient condition on domination rules which is satisfied by all example rules that we mentioned and which implies (conditional) intractability (Theorem 7.5). We then show that, for a class of domination rules called *variable-inclusion-like* rules, a variant of this condition can be used for a dichotomy to classify which of these rules enjoy tractable skyline computation (Theorem 7.7). We finish with examples of tractable and intractable rules in the general case.

We conclude in Section 8. For reasons of space, most proofs are deferred to the full version [4].

## 2 Preliminaries

**Languages, spans, mappings, and spanners.** We fix an *alphabet*  $\Sigma$  which is a finite set of letters. A *word*  $w$  is a finite sequence of letters of  $\Sigma$ : we write  $\Sigma^*$  for the set of all words. We write  $|w|$  for the length of  $w$  and denote the empty word by  $\varepsilon$ , with  $|\varepsilon| = 0$ . A *language*  $L \subseteq \Sigma^*$  is a set of words. The *concatenation* of two languages  $L_1$  and  $L_2$  is the language  $L_1 \cdot L_2 = \{w_1w_2 \mid w_1 \in L_1, w_2 \in L_2\}$ . The *Kleene star* of a language  $L$  is the language  $L^* = \bigcup_{i \in \mathbb{N}} L^i$ , where we define inductively  $L^0 = \{\varepsilon\}$  and  $L^{i+1} = L \cdot L^i$  for all  $i > 0$ . As usual in the context of document spanners, a *document* is simply a word of  $\Sigma^*$ .

A *span*  $[i, j\rangle$  is an interval  $s = [i, j\rangle$  with  $0 \leq i \leq j$ . Its *length* is  $j - i$ . We denote by **Spans** the set of all spans. The *spans* of a document  $d$  are the spans  $[i, j\rangle$  of **Spans** with  $j \leq |d|$ . We write  $d_{[i, j\rangle}$  to mean the contiguous subword of  $d$  at a span  $[i, j\rangle$ , for example “qwertyqwerty” $_{[2,5\rangle} =$  “qwertyqwerty” $_{[8,11\rangle} =$  “ert”. Note that we have  $d_{[i, i\rangle} = \varepsilon$  for all  $0 \leq i \leq |d|$ . A span  $[i, j\rangle$  is *included* in a span  $[i', j'\rangle$  if  $i' \leq i$  and  $j' \geq j$ . Two spans *overlap* if there is a non-empty span included in both of them; otherwise we call them *disjoint*.

## 7:4 Skyline Operators for Document Spanners

We fix an infinite set  $\text{Variables}$  of variable names. A *mapping*  $m$  of a document  $d \in \Sigma^*$  is a function from a finite set of variables  $X \subseteq \text{Variables}$ , called the *domain*  $\text{dom}(m)$  of  $m$ , to the set of spans of  $d$ ; the variables of  $\text{dom}(m)$  are said to be *assigned* by  $m$ . We denote the set of all mappings on variables of  $\text{Variables}$  by  $\text{Maps}$ , and denote by  $\text{Maps}(d)$  the set of all mappings on a document  $d$ , i.e., the mappings of  $\text{Maps}$  such that all spans in the image are spans of  $d$ . A mapping  $m$  is called *compatible* with a mapping  $m'$ , in symbols  $m \sim m'$ , if for all  $x \in \text{dom}(m) \cap \text{dom}(m')$  we have  $m(x) = m'(x)$ .

A *spanner* is a function mapping every document  $d$  to a finite set of mappings whose spans are over  $d$ , i.e., are included in  $[0, |d|]$ . For a spanner  $P$ , we denote by  $\text{SVars}(P)$  the variables appearing in the domain of at least one of its mappings, formally  $\text{SVars}(P) := \{x \in \text{Variables} \mid \exists d \in \Sigma^*, \exists m \in P(d), x \in \text{dom}(m)\}$ . A spanner  $P$  is *schema-based* if all its output mappings assign exactly the variables of  $\text{SVars}(P)$ , i.e., for every  $d \in \Sigma^*$  and  $m \in P(d)$ , we have  $\text{dom}(m) = \text{SVars}(P)$ . Otherwise,  $P$  is called *schemaless* [21], or *incomplete* [18]. We say a spanner  $P$  *accepts* or *captures* a mapping  $m \in \text{Maps}$  on a document  $d \in \Sigma^*$  if  $m \in P(d)$ .

**Variable-set automata.** We focus mostly on the *regular spanners*, that can be expressed using *variable-set automata* (or VAs). These are intuitively nondeterministic automata where each transition is labeled either by a letter or by a *marker* indicating which variable is opened or closed. Formally, for a set  $X$  of variables, we denote by  $\text{markers}(X)$  the set of *markers* over  $X$ : it contains for each variable  $x \in X$  the *opening marker*  $x\vdash$  and the *closing marker*  $\dashv x$ . Then, a VA on alphabet  $\Sigma$  is an automaton  $\mathcal{A} = (Q, q_0, F, \delta)$  where  $Q$  is a finite set of *states*,  $q_0 \in Q$  is the *initial state*,  $F \subseteq Q$  are the *final states*, and  $\delta \subseteq Q \times (\Sigma \cup \text{markers}(X)) \times Q$  is the *transition relation*: we write the transitions  $q \xrightarrow{\sigma} q'$  to mean that  $(q, \sigma, q') \in \delta$ . Note that the transitions contain both *letter transitions*, labeled by letters of  $\Sigma$ , and *marker transitions*, labeled by markers of  $\text{markers}(X)$ .

A *run* of  $\mathcal{A}$  on a document  $d \in \Sigma^*$  is a sequence  $\rho : q_0 \xrightarrow{\sigma_1} q_1 \cdots q_{n-1} \xrightarrow{\sigma_n} q_n$  such that the restriction of  $\sigma_1 \dots \sigma_n$  to the letters of  $\Sigma$  is exactly  $d$ ; it is *accepting* if we have  $q_n \in F$ . We say that  $\rho$  is *valid* if, for each variable  $x \in X$ , either the markers  $x\vdash$  and  $\dashv x$  do not occur in  $\sigma_1 \cdots \sigma_n$ , or they occur exactly once and  $x\vdash$  occurs before  $\dashv x$ . We say that  $\mathcal{A}$  is *sequential* if all its accepting runs are valid. In this paper, we always assume that VAs are sequential, and only speak of VAs to mean sequential VAs. The run  $\rho$  then defines a mapping  $m$  on  $d$  by intuitively assigning the variables for which markers are read to the span delimited by these markers. Formally, we associate to each index  $0 \leq k \leq n$  of the run a position  $\pi(k)$  in  $d$  by initializing  $\pi(0) := 0$  and setting  $\pi(k+1) := \pi(k)$  if the transition  $q_k \xrightarrow{\sigma_{k+1}} q_{k+1}$  reads a marker, and  $\pi(k+1) := \pi(k) + 1$  if it reads a letter; note that  $\pi(n) = |d|$ . Then, for each variable  $x$  whose markers are read in  $\rho$ , letting  $\sigma_i = x\vdash$  and  $\sigma_j = \dashv x$  with  $i < j$  because the run is valid, we set  $m(x) := [\pi(i), \pi(j)]$ .

A sequential VA  $\mathcal{A}$  thus defines a spanner  $P_{\mathcal{A}}$  that maps each document  $d$  to the set  $P_{\mathcal{A}}(d)$  of mappings obtained from its accepting runs as we explained. Note that different accepting runs may yield the same mapping. We sometimes abuse notation and identify VAs with the spanners that they define. The *regular spanners* are those that can be defined by VAs, or, equivalently [18], by sequential VAs. A sequential VA is *functional* if it defines a schema-based spanner, i.e., every mapping assigns every variable that occurs in the transitions of the VA.

**Regex formulas.** Our examples of spanners in this paper will be given not as VAs but in the more human-readable formalism of *regex formulas*. The *regex formulas* over an alphabet  $\Sigma$  are the expressions defined inductively from the empty set  $\emptyset$ , empty word  $\varepsilon$ , and single

letters  $a \in \Sigma$ , using the three regular operators of disjunction ( $e_1 \vee e_2$ ), concatenation ( $e_1 e_2$ ), and Kleene star ( $e^*$ ), along with *variable captures* of the form  $x\{e_1\}$  where  $x$  is a variable. A regex-formula  $r$  on a document  $d \in \Sigma^*$  defines a spanner on the variables occurring in  $r$ . Intuitively, every match of  $r$  on  $d$  yields a mapping where the variables are assigned to well-nested spans following the captures; see [10] for details. We require of regex-formulas that, on every document  $d \in \Sigma^*$ , they assign each variable at most once; but we allow them to define schemaless spanners, i.e., they may only assign a subset of the variables.

It is known that regex formulas capture a strict subset of the regular spanners; see [10] in the case of schema-based spanners and [18] for the case of schemaless spanners.

**Cartesian Products.** Given two spanners  $P_1$  and  $P_2$  where  $X_1 = \text{SVars}(P_1)$  and  $X_2 = \text{SVars}(P_2)$  are disjoint, the *Cartesian product*  $P_1 \times P_2$  of  $P_1$  and  $P_2$  is the spanner on variables  $X_1 \cup X_2$  which on every document  $d$  captures the mappings  $(P_1 \times P_2)(d) := P_1(d) \times P_2(d)$ . Here, we interpret a pair  $(m_1, m_2) \in P_1(d) \times P_2(d)$  as the merge of the two mappings, i.e., the mapping defined according to  $m_1$  on  $X_1$  and according to  $m_2$  on  $X_2$ . If  $P_1$  and  $P_2$  are given as sequential VAs, then one can compute in polynomial time a sequential VA for  $P_1 \times P_2$ .

### 3 The Skyline Operator

In this paper, we define and study a new operator called the *skyline operator*. Its goal is to only extract mappings that contain the maximum amount of information in a certain sense.

**Domination relations.** We begin by defining *domination relations* which describe how to compare the information given by two mappings on a given document  $d$ .

► **Definition 3.1.** A pre-domination relation  $\preceq$  is a binary relation on the set of mappings  $\text{Maps}(d)$  of  $d$ . We say that it is a domination relation if it is a (non-strict) partial order, i.e., it is reflexive, transitive, and antisymmetric. For  $m_1, m_2 \in \text{Maps}$ , we say that  $m_2$  dominates  $m_1$  if  $m_1 \preceq m_2$ , and write  $m_1 \not\preceq m_2$  otherwise.

The goal of the domination relation is to define which mappings are preferred to others, intuitively because they contain more information; it may depend on the document, though we will present many examples where it does not.

We introduce several domination relations that, as discussed in the introduction, are part of practical systems and which we consider throughout this paper:

► **Definition 3.2.** The simplest relation is the trivial self domination relation  $\preceq_{\text{self}}$  where every mapping only dominates itself, i.e., the pairs in the relation are  $(m, m)$  for  $m \in \text{Maps}$ .

► **Definition 3.3.** The variable inclusion relation  $\preceq_{\text{varInc}}$  contains the pairs  $(m_1, m_2)$  such that for all  $x \in \text{Variables}$ , if  $m_1(x)$  is defined, then  $m_2(x)$  is defined as well and  $m_1(x) = m_2(x)$ . Thus, we have  $m_1 \preceq_{\text{varInc}} m_2$  whenever  $\text{dom}(m_1) \subseteq \text{dom}(m_2)$  and  $m_1 \sim m_2$ , i.e., when  $m_2$  is an extension of  $m_1$  that potentially assigns more variables than  $m_1$ .

► **Definition 3.4.** The span inclusion relation  $\preceq_{\text{spanInc}}$  contains the pairs  $(m_1, m_2)$  of mappings with the same domain ( $\text{dom}(m_1) = \text{dom}(m_2)$ ) such that for every  $x \in \text{dom}(m_1)$  the span  $m_1(x)$  is included in  $m_2(x)$ . Intuitively,  $m_1$  and  $m_2$  match the same variables in the same parts of a document, but the matches of variables in  $m_1$  are subwords of their matches in  $m_2$ .

► **Definition 3.5.** The left-to-right relation  $\leq_{ltr}$  contains the pairs  $(m_1, m_2)$  of mappings with the same domain such that, for every variable  $x$  on which  $m_1$  and  $m_2$  are defined, the spans  $m_1(x)$  and  $m_2(x)$  start at the same position but  $m_2(x)$  is no shorter than  $m_1(x)$ .

► **Definition 3.6.** The span length relation  $\leq_{spanLen}$  contains the pairs  $(m_1, m_2)$  of mappings with the same domain where for every  $x \in \text{dom}(m_1)$  the span  $m_2(x)$  is no shorter than  $m_1(x)$ . Intuitively,  $\leq_{spanLen}$  prefers longer spans over shorter ones, anywhere in the document.

**Domination rules.** We now introduce *domination rules* which associate to each document  $d$  a domination relation over  $d$ . In this paper, we express domination rules as spanners on specific domains. To this end, given a set of variables  $X$ , we write  $X^\dagger$  to mean a set of annotated copies of the variables of  $X$ , formally  $X^\dagger := \{x^\dagger \mid x \in X\}$ . We extend the notation to mappings by defining  $m^\dagger$  for a mapping  $m$  to be the mapping with domain  $\text{dom}(m^\dagger) = \text{dom}(m)^\dagger$  such that for all  $x \in \text{dom}(m)$  we have  $m^\dagger(x^\dagger) := m(x)$ . We then define:

► **Definition 3.7.** A pre-domination rule  $D$  on a set of variables  $X \subseteq \text{Variables}$  is a (schemaless) spanner with  $\text{SVars}(D) \subseteq X \cup X^\dagger$ . For every document  $d \in \Sigma^*$ , we see  $D(d)$  as a pre-domination relation  $\leq$  on  $d$  defined by the mappings captured by  $D$  on  $d$ , the left-hand-side and right-hand-side of the comparability pairs being the restrictions of the mappings to  $X$  and to  $X^\dagger$  respectively. Formally, the relation  $\leq$  is:  $R := \{(m|_X, m') \mid m \in D(d), (m')^\dagger = m|_{X^\dagger}\}$ .

We say that  $D$  is a domination rule if, on every document  $d \in \Sigma^*$ , the pre-domination relation  $R$  defined above is a domination relation, i.e., it correctly defines a partial order.

Intuitively, for every document  $d$ , the domination rule  $D$  defines the domination relation  $\leq$  where each mapping  $m \in D(d)$  denotes a pair, i.e., the restriction of  $m$  to  $X$  is dominated by the restriction of  $m$  to  $X^\dagger$  (renaming the variables from  $X^\dagger$  to  $X$ ). Note that pre-domination rules and pre-domination relations are just an intermediary notion; in the sequel, we only consider domination rules and domination relations.

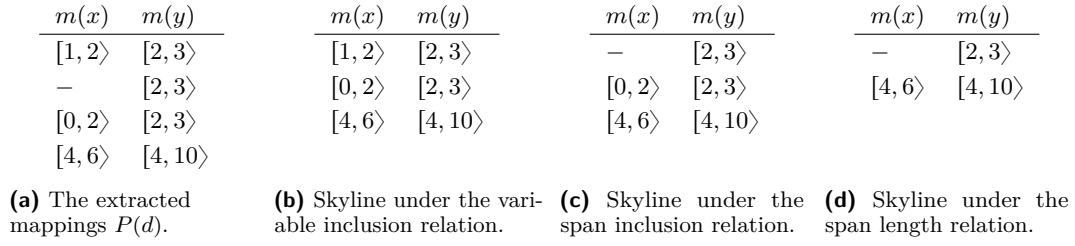
► **Example 3.8.** For any set  $X$  of variables, each of the domination relations introduced in Definitions 3.2–3.5 can be defined by a domination rule expressed by a regular spanner on  $X$  (for the span length domination relation of Definition 3.6, see Lemma 3.13). At the end of the section, we explain how to express them in a more concise *variable-wise* way that does not depend on  $X$ : see Example 3.12.

**The skyline operator.** We have introduced domination rules as a way to define domination relations over arbitrary documents. We can now introduce the *skyline operator* to extract maximal mappings, i.e., mappings that are not dominated in the domination relation:

► **Definition 3.9.** Given a domination rule  $D$ , the skyline operator  $\eta_D$  of  $D$  applies to a spanner  $P$  and defines a spanner  $\eta_D P$  in the following way: given a document  $d$ , writing  $\leq$  to denote the domination relation  $D(d)$  given by  $D$  on  $d$ , the result of  $\eta_D P$  on  $d$  is the set of maximal mappings of  $P(d)$  under the domination relation  $\leq$ . Formally, we have:  $(\eta_D P)(d) := \{m \in P(d) \mid \forall m' \in P(d) \setminus \{m\} : m \not\leq m'\}$ .

Intuitively, the operator filters out the mappings that are dominated by another mapping according to the domination relation defined by the domination rule over the input document.

► **Example 3.10.** In Figure 1 we show the effect of the skyline operator with respect to some of our example domination relations. Assume that we are given a spanner  $P$  in variables  $\{x, y\}$  that on a given document  $d$  extracts the mappings given in Figure 1a (here a dash “–”



■ **Figure 1** Extracted mappings before and applying different skyline operators; see Example 3.10.

means that the variable is not assigned by a mapping). We show the result of applying the skyline operators with (possibly non-regular) domination rules defining the variable inclusion domination relation  $\leq_{varInc}$  (Figure 1b), the span inclusion domination relation  $\leq_{spanInc}$  (Figure 1c), and the span length domination relation  $\leq_{spanLen}$  (Figure 1d). Note that, for the variable inclusion domination rule, the skyline only makes sense for schemaless spanners, as two distinct mappings that assign the same variables are always incomparable.

**Variable-wise rules.** We have defined our skyline operator relative to domination rules expressed as spanners on explicit sets of variables. However, it will often be convenient to define the rules as products of rules on a single variable by applying the product operator. This ensures that the rule is “symmetric” in the sense that all variables behave the same:

► **Definition 3.11.** *Let  $D$  be a domination rule in a single variable  $x$ , i.e., a spanner using variables of  $\{x, x^\dagger\}$ , which we call a single-variable domination rule. For  $y \in \text{Variables}$ , we let  $D^y$  be the domination rule where we replace  $x$  and  $x^\dagger$  by  $y$  and  $y^\dagger$ , i.e., on every document  $d$ , the set of mappings  $D^y(d)$  consists of one mapping  $m^y$  for each mapping  $m \in D(d)$  with  $m^y(y)$  and  $m^y(y^\dagger)$  defined like  $m(x)$  and  $m(x^\dagger)$ .*

*The variable-wise domination rule defined by  $D$  on a variable set  $X$  is then simply  $\times_{y \in X} D^y$ . A domination rule is said to be variable-wise if it can be expressed in this way.*

We will often leave the set of variables  $X$  implicit, and may abuse notation to identify single-variable domination rules with the variable-wise domination rule that they can define on an arbitrary variable set.

► **Example 3.12.** The self domination rule (Definition 3.2) is variable-wise, because it can be obtained from the following trivial single-variable domination rule:

$$D_{self} = \Sigma^* x^\dagger \{x \{ \Sigma^* \} \} \Sigma^* \vee \Sigma^*.$$

The  $\Sigma^*$  term above is used to ensure reflexivity and express the vacuous domination relation between the mapping where  $x$  is not assigned and the mapping where  $x^\dagger$  is not assigned.

The span inclusion domination rule, left-to-right domination rule, and variable inclusion domination rule (Definitions 3.3–3.5) are also variable-wise with the single-variable rules:

$$D_{spanInc} = \Sigma^* x^\dagger \{ \Sigma^* x \{ \Sigma^* \} \Sigma^* \} \Sigma^* \vee \Sigma^*.$$

$$D_{ltr} = \Sigma^* x^\dagger \{ x \{ \Sigma^* \} \Sigma^* \} \Sigma^* \vee \Sigma^*.$$

$$D_{varInc} = \Sigma^* x^\dagger \{ \Sigma^* \} \Sigma^* \vee D_{self}.$$

Here,  $\Sigma^* x^\dagger \{ \Sigma^* \} \Sigma^*$  expresses that assigning a variable is better than not assigning it.

As for the span length domination rule (Definition 3.6), it is also variable-wise, but a standard pumping argument shows that it cannot be defined by a regular spanner:

► **Lemma 3.13.** *The single-variable span length domination rule  $D_{spanLen}$  is not expressible as a regular spanner.*



## 4 Closure under the Skyline Operator

We have defined the skyline operator relative to domination rules expressed by regular spanners. One natural question is then to understand whether the skyline operator under such rules extends the expressive power of spanner formalisms, or whether it can be defined in existing models. This is what we investigate in this section.

**Regular spanners.** We first focus on regular spanners, and show that they are closed under the skyline operator for domination rules expressed as regular spanners. We do so by showing how the skyline operator can be expressed with operations under which regular spanners are closed, namely join, intersection and difference (see Appendix A for definitions).

► **Theorem 4.1.** *There is an algorithm that, given a sequential VA defining a regular spanner  $P$  and a sequential VA defining a domination rule  $D$ , computes a sequential VA for  $\eta_D P$ .*

Theorem 4.1 generalizes a result of Fagin et al. [11, Theorem 5.3] on the expressiveness of transitive “denial pgds.” In our terminology, their theorem states that the class of *complete* regular spanners is closed under the restriction to maximal answers defined by a regular domination rule. Theorem 4.1 thus extends their result to schemaless regular spanners.

Theorem 4.1 implies that taking the skyline relative to regular domination rules does not increase the expressivity of regular spanners. However, like the result of [11], our construction may compute VAs that are exponentially bigger than the input VA. In Section 5, we will see that this is unavoidable for any sequential VA expressing the skyline.

As an application of Theorem 4.1 we get in particular that regular spanners are closed under the skyline operator for most of the examples presented earlier, i.e., Definitions 3.2–3.5.

► **Corollary 4.2.** *There are algorithms that, given a sequential VA  $P$ , compute sequential VAs for  $\eta_{self} P$ ,  $\eta_{varInc} P$ ,  $\eta_{tr} P$ , and  $\eta_{spanInc} P$ , respectively.*

By contrast, Theorem 4.1 does not apply to the span-length domination rule, as it is not expressible as a regular spanner (Lemma 3.13). In fact, we can show that taking the skyline under this domination rule is generally *not* expressible as a regular spanner:

► **Proposition 4.3.** *There is a sequential VA  $P$  such that  $\eta_{spanLen} P$  is not regular.*

**Other spanner formalisms.** It is natural to ask whether closure results such as Theorem 4.1 also hold for other spanner formalisms. In particular, we can ask this for the language of *core spanners*, which extend regular spanners with string equalities; see [10] for the precise definitions and [24] for the schemaless case. We can show that core spanners, contrary to regular spanners, are *not* closed under the skyline operator:

► **Theorem 4.4.** *The core spanners are not closed under the skyline operator with respect to the span inclusion domination rule  $D_{spanInc}$ , even on schema-based spanners: there is a schema-based core spanner  $P$  such that  $\eta_{spanInc} P$  cannot be expressed as a core spanner. The same is true of the skyline  $\eta_{varInc}$  with the variable inclusion domination rule.*

This result was already shown in [11] for the span inclusion domination rule, but that result only showed inexpressibility as a schema-based core spanner. Our result extends to the schemaless setting, and also establishes the result for the variable inclusion domination rule. See the full version [4] for the formal definitions and the proof.

We leave open the question of extending other formalisms with the skyline operator, e.g., the *generalized core spanners* which extend core spanners with the difference operator [22], or the *context-free spanners* [20] that define spanners via context-free grammars. Note that, by



contrast, closure is easily seen to hold in the formalism of *RGXlog* programs, where spanners are defined using Datalog rules [22]. Indeed, this class consists of precisely the polynomial-time spanners (under data complexity). Thus, for any domination rule  $D$  for which the maximal answers can be computed in polynomial time data complexity (in particular, for domination rules expressed as regular spanners), the result of the skyline operator for  $D$  on an *RGXlog* program can be expressed as an *RGXlog* program.

In the rest of this paper, we focus on applying the skyline operators to regular spanners, with domination relations also defined via regular domination rules.

## 5 State Complexity of the Skyline Operator

We have seen how the skyline operator does not increase the expressive power of regular spanners, in the sense that it could be expressed using regular operations. However, this does not account for the price of this transformation. In this section, we show that the size of sequential VAs for some domination rules increases exponentially when applying the skyline operator. Hence, we essentially study the state complexity of VAs under the skyline operator, similarly to traditional studies of the state complexity for regular languages (e.g., [15]). Specifically, we show the following lower bound, for the variable inclusion domination rule:

► **Theorem 5.1.** *For every  $n \in \mathbb{N}$ , there is a sequential VA  $\mathcal{A}$  with  $O(n)$  states such that, letting  $P_{\mathcal{A}}$  be the regular spanner that it defines, any sequential VA representing the regular spanner  $\eta_{\text{varInc}}P_{\mathcal{A}}$  must have  $2^{\Omega(n)}$  states.*

We will show in later sections how this lower bound on the state complexity of the skyline operation can be complemented with computational complexity lower bounds.

**Proof technique: Representing Boolean functions as VAs.** We show Theorem 5.1 using representations of Boolean functions as sequential VAs, as we now explain. Let  $\text{SVars} \subseteq \text{Variables}$  be a finite set of variables (which will be used to define spanners), and let  $\text{Vars}_b := \{x_b \mid x \in \text{SVars}\}$  be a set of Boolean variables. For every mapping  $m$  assigning spans to some of the variables in  $\text{SVars}$  (i.e.,  $\text{dom}(m) \subseteq \text{SVars}$ ), we define a Boolean assignment  $m_b: \text{Vars}_b \rightarrow \{0, 1\}$  by setting  $m_b(x_b) := 1$  if and only if  $x \in \text{dom}(m)$ , i.e.,  $x$  gets assigned a span by  $m$ . Let  $P$  be a document spanner with variables  $\text{SVars}$  and let  $d$  be an input document. Then we denote by  $\text{Bool}(P, d)$  the Boolean function whose models are  $\{m_b \mid m \in P(d)\}$ .

Our intuitive idea is that, if the function  $\text{Bool}(P, d)$  is hard to represent, then the same should be true of the spanner  $P$ . To make this precise, let us introduce the representations of Boolean functions that we work with:

► **Definition 5.2.** *A nondeterministic read-once branching program<sup>1</sup> (NROBP) over the variable set  $\text{Vars}_b$  is a tuple  $\Pi = (G, s, t, \mu)$  where  $G = (V, E)$  is a directed acyclic graph,  $s \in V$  and  $t \in V$  are respectively the source and sink nodes, and the function  $\mu$  labels some of the edges with literals of variables in  $\text{Vars}_b$ , i.e., variables and their negations; formally  $\mu$  is a partial function from  $E$  to the literals over  $\text{Vars}_b$ . We require that, for every source-sink path  $s = v_0, \dots, v_n = t$ , every variable appears at most once in the literals labeling the edges of the path, i.e., there are no two indices  $0 \leq i < j \leq n - 1$  such that  $\mu((v_i, v_{i+1}))$  and  $\mu((v_j, v_{j+1}))$  are both defined and map to literals of the same variable.*

<sup>1</sup> We remark that what we introduce here are sometimes called *acyclic read-once switching and rectifier networks*, but these are known to be equivalent to the more common definition of NROBPs up to constant factors [23], so we do not make the difference here.

An NROBP  $\Pi$  computes a Boolean function over  $\text{Vars}_b$  whose models are defined in the following way. An assignment  $m_b: \text{Vars}_b \rightarrow \{0, 1\}$  is a model of  $\Pi$  if there is a source-sink path in  $G$  such that all literal labels on the path are satisfied by  $m_b$ , i.e., there is a sequence  $s = v_0, \dots, v_n = t$  such that, for each  $0 \leq i < n$  for which  $\ell := \mu((v_i, v_{i+1}))$  is defined, then the literal  $\ell$  evaluates to true according to  $m_b$ .

NROBPs are intuitively similar to automata. To formalize this connection, we show how, given a sequential VA and document, we can efficiently compute an NROBP describing which subsets of the variables can be assigned in captured mappings:

► **Lemma 5.3.** *Let  $P$  be a regular spanner on variable set  $\text{SVars}$  represented by a sequential VA  $\mathcal{A}$  with  $n$  states. Then, for every document  $d$ , there is an NROBP  $G$  representing  $\text{Bool}(P, d)$  with  $O(|d| \times n \times |\text{SVars}|)$  nodes.*

**Proof sketch.** We compute the product of the VA with the input document, to obtain a directed acyclic graph representing the runs of the VA on the document. We obtain the NROBP by relabeling the marker transitions and performing some other modifications. ◀

We will now use the fact that NROBPs are exponentially less concise than other Boolean function representations. Namely, we define a *read-3 monotone 2-CNF formula* on a set of variables  $X$  as a conjunction of clauses which are disjunctions of 2 variables from  $X$ , where each variable appears at most 3 times overall. We use the fact that converting such formulas to NROBPs can incur an exponential blowup. This result is known (see, e.g., [7]) but we give a proof in the full version [4] for convenience:

► **Proposition 5.4** ([7]). *For any  $n \in \mathbb{N}$ , there is a read-3 monotone 2-CNF formula  $\Phi$  on  $n$  variables having size  $O(n)$  such that every representation of  $\Phi$  as an NROBP has size  $2^{\Omega(n)}$ .*

We now conclude the proof of Theorem 5.1, sketched below (see the full version [4] for details):

**Proof sketch.** Given a read-3 monotone 2-CNF formula  $\Phi$ , we show how to construct a regular spanner on which the skyline operator captures mappings corresponding precisely to the satisfying assignments of  $\Phi$ . As a sequential VA expressing this spanner can be efficiently converted to an NROBP by Lemma 5.3, we can conclude that, when applied to the family of formulas from Proposition 5.4, all sequential VA representations have exponential size. ◀

#### **An independent result: Lower bound on the state complexity of schema-less joins.**

We believe that the connection to Boolean functions used to show Theorem 5.1 can be of independent interest as a general technique to show lower bounds on the state complexity of document spanners. Indeed, independently from the skyline operator, we can also use this connection to show a lower bound on the size of sequential VAs representing the *natural join* of two regex-formulas. The *natural join operator* is a standard operator on spanners that merges together compatible mappings: see Appendix A for the formal definition. We have:

► **Theorem 5.5.** *For every  $n \in \mathbb{N}$ , there are regex-formulas  $e_n$  and  $e'_n$  of size  $O(n)$  such that every sequential VA equivalent to  $e_n \bowtie e'_n$  has  $2^{\Omega(n)}$  states.*

This result is the counterpart for state complexity of the NP-hardness of evaluating the join of two regex-formulas [21]. It only holds in the schemaless case; indeed in the schema-based case it is known that the join of two functional VAs can be computed as a functional VA in polynomial time [13].

## 6 Complexity of the Skyline Operator

We have shown that the skyline operator applied to regular spanners cannot be expressed as a regular spanner without an exponential blowup in the size, even for domination rules expressed as regular spanners (namely, for the variable inclusion domination rule). We now study whether we can efficiently evaluate the skyline operator without compiling it into the automaton. Formally, we study its computational complexity of skyline extraction:

► **Definition 6.1.** *The skyline extraction problem is the following: given a document  $d$ , a sequential VA  $\mathcal{A}$  capturing a regular spanner  $P_{\mathcal{A}}$ , and a domination rule  $D$  expressed as a sequential VA, compute the set of mappings in the results of the skyline operator  $(\eta_D P_{\mathcal{A}})(d)$ .*

**Data complexity.** We start by observing that skyline extraction is clearly tractable in the data complexity perspective in which  $d$  is the only input:

► **Proposition 6.2.** *For any fixed sequential VA  $\mathcal{A}$  and domination rule expressed as a sequential VA  $D$ , the skyline extraction problem for  $P_{\mathcal{A}}$  and  $D$  can be solved in polynomial time data complexity, i.e., in polynomial time in the input  $d$ .*

**Proof.** We use Theorem 4.1 to compute a sequential VA for  $(\eta_D P_{\mathcal{A}})(d)$ , which is independent from the input document  $d$ . We can then produce the mappings captured by this fixed sequential VA on  $d$  in polynomial data complexity.

Alternatively, without using the theorem, note that we can simply materialize the set of all captured mappings  $(P_{\mathcal{A}})(d)$ , in polynomial time because  $\mathcal{A}$  is fixed. Then, for any pair of mappings, we can check if the domination relation holds using the domination rule  $D$ ; this is again in polynomial time. We then return the set of maximal mappings in polynomial time. ◀

Note that this result would easily extend to fixed expressions using multiple skyline operators together with regular spanner operators, as all these operators are polynomial-time.

**Combined complexity.** We now turn to combined complexity settings in which the domination rule  $D$  and the spanner  $P$  are considered as part of the input. In fact, we will mostly consider the problem variant in which we fix a single-variable domination rule (e.g., variable inclusion), we take the skyline relative to the corresponding variable-wise domination rule, and only the spanner  $P$  is part of the input. Remember that we focus on regular spanners represented as sequential VAs, since for those it is known that the combined complexity of spanner evaluation is output polynomial [18].

As we have seen in Section 4, in terms of expressiveness, the regular spanners are closed under all domination rules expressible as regular spanners, in particular those of Definitions 3.2–3.5. However, we have seen in Section 5 that compiling the skyline into the VA may generally incur an exponential blowup, already for fixed domination rules. This bars any hope of showing tractability of the skyline extraction problem by applying known evaluation algorithms on the result of this transformation (e.g., those from [13, 12, 3]),

This leads to the question if there are other approaches to solve the skyline extraction problem with efficient combined complexity, without materializing an equivalent VA. In this section, we show that this is not the case, assuming  $P \neq NP$ . Our lower bound already holds for a fixed domination rule, namely, the variable inclusion domination rule; and in fact it even holds in *query complexity*, i.e., when the document is fixed.

► **Theorem 6.3.** *There is a fixed document  $d$  such that the following problem is NP-hard: given a sequential VA  $\mathcal{A}$  encoding a regular spanner  $P_{\mathcal{A}}$  and a number  $n \in \mathbb{N}$  which is at most the size of  $\mathcal{A}$ , decide whether  $(\eta_{\text{varInc}}P_{\mathcal{A}})(d)$  contains more than  $n$  mappings.*

This will imply that, conditionally, the skyline extraction problem is intractable in combined complexity. Intuitively, if it is intractable to decide whether the skyline extracts a large number of mappings, then producing the mappings is also intractable. To make this formal, we use the framework of *output-polynomial algorithms*, where an algorithm for a problem  $f: \Sigma^* \rightarrow \Sigma^*$  runs in *output-polynomial time* if, given an input  $x$ , it runs in time polynomial in  $|x| + |f(x)|$ . Namely, we use the following folklore connection between output-polynomial time and decision problems, see e.g. [8] for a similar construction:

► **Lemma 6.4.** *Let  $f: \Sigma^* \rightarrow \Sigma^*$  and let  $p$  be a polynomial. Assume that it is NP-hard, given an input  $x$  and integer  $k \leq p(|x|)$ , to decide if  $|f(x)| < k$ . Then there is no output polynomial time algorithm for  $f$ , unless  $P = NP$ .*

From Theorem 6.3 and Lemma 6.4, we directly get our intractability result:

► **Corollary 6.5.** *Unless  $P = NP$ , there is no algorithm for the skyline extraction problem with respect to the variable inclusion domination rule that is output-polynomial in combined complexity (i.e., in the input sequential VA), even when the input document is fixed.*

Note that this result is incomparable to Theorem 5.1: lower bounds on the size of equivalent VAs generally do not preclude the existence of other algorithms that are tractable in combined complexity, and conversely it could in principle be the case that evaluation is intractable in combined complexity but that there are small equivalent VAs that are intractable to compute. Besides, the proofs are also different. Namely, the proof of Theorem 5.1 used monotone 2-CNF formulas, for which we could compute spanners giving an exact representation of the satisfying assignments, but for which the satisfiability problem is tractable. As we will see, the proof of Theorem 6.3 uses the intractability of SAT on CNF formulas, but does not use an exact representation of the satisfying assignments.

**Proving Theorem 6.3.** We give the proof of Theorem 6.3 in the rest of this section, together with an additional observation at the end. In the next section, we will study how hardness can be generalized to other domination rules (in particular all domination rules introduced in Section 3 except the trivial self-domination rule), and will investigate the existence of tractable cases.

**Proof of Theorem 6.3.** We reduce from the satisfiability problem SAT. Let  $F$  be a CNF formula with  $n_x$  Boolean variables  $x_i$  with  $i \in [n_x]$  and  $n_c$  clauses  $C_j$  with  $j \in [n_c]$ . For convenience, define the set  $T_i = \{j \mid x_i \text{ appears positively in } C_j\}$ , and define the set  $F_i = \{j \mid x_i \text{ appears negatively in } C_j\}$ . We will build a regular spanner on variables  $v_{i,j}$  for  $i \in [n_x]$  and  $j \in [n_c]$ , together with a special variable  $a$ .

We will define two spanners  $r_{\text{valid}}$  and  $r_{\text{mask}}$ , both as regex formulas, and will evaluate them on the empty document  $d = \varepsilon$ . Let us first sketch the idea: the spanner  $r_{\text{valid}}$  will extract one mapping for each possible assignment to the variables of  $F$ . Each such mapping will encode which clauses get satisfied by which variable in the assignment, by assigning spans to the corresponding spanner variables  $v_{i,j}$ . The second spanner  $r_{\text{mask}}$  will capture  $n_c$  additional mappings which will be maximal (thanks to the additional variable  $a$ ) and will each dominate the mappings captured by  $r_{\text{valid}}$  for which the corresponding assignment does *not* satisfy a specific clause of  $F$ . This will ensure that  $F$  is satisfiable if and only if there are strictly more than  $n_c$  mappings in the skyline of  $r_{\text{valid}} \vee r_{\text{mask}}$  on  $d$ .

Formally, we define the spanners as regex-formulas, where the dots denote concatenation:

$$r_{\text{valid}} = \cdot_{i \in [n_x]} ((\cdot_{j \in T_i} v_{i,j} \{\varepsilon\}) \vee (\cdot_{j \in F_i} v_{i,j} \{\varepsilon\})) \quad r_{\text{mask}} = a\{\varepsilon\} \cdot \bigvee_{k \in [n_c]} \cdot_{i \in [n_x], j \in [n_c] \setminus \{k\}} v_{i,j} \{\varepsilon\}.$$

This definition is in polynomial time in the input CNF  $F$ .

Note that the mappings captured by  $r_{\text{mask}}$  are never dominated. First, they do not dominate each other: each of them assigns no  $v_{i,k}$  for some  $k$ . Further, all mappings of  $r_{\text{mask}}$  assign  $a$  and all mappings of  $r_{\text{valid}}$  do not, so the latter cannot dominate the former.

To construct a CNF variable assignment from a mapping  $m$  captured by  $r_{\text{valid}}$ , we use the following encoding: if the mapping  $m$  assigns the span  $[0, 0\rangle$  to the spanner variable  $v_{i,j}$  then this encodes that the variable  $x_i$  appears in the clause  $C_j$  and  $x_i$  is assigned in a way that satisfies  $C_j$ . The definition of  $r_{\text{valid}}$  ensures that all variables appearing at least once will be assigned exactly one truth value among true or false.

We claim that on  $d = \varepsilon$ , the skyline  $(\eta_{\text{varInc}}(r_{\text{valid}} \vee r_{\text{mask}}))(d)$  contains at least  $n_c + 1$  mappings if and only if  $F$  is satisfiable. Assume first that  $F$  is satisfiable, and let  $v$  be a satisfying assignment. Then there is a corresponding mapping  $m$  captured by  $r_{\text{valid}}$  encoding  $v$ . Indeed, as  $v$  satisfies all clauses, for every clause index  $j \in [n_c]$  there is a variable  $x_i$  assigned by  $v$  in a way that makes  $C_j$  true, i.e.,  $v_{i,j}$  is assigned. Hence  $m$  will not be dominated by any mapping captured by  $r_{\text{mask}}$ . Thus, the skyline of  $r_{\text{valid}} \vee r_{\text{mask}}$  must contain some mapping captured by  $r_{\text{valid}}$ , namely, either  $m$  or some other mapping captured by  $r_{\text{valid}}$  which dominates  $m$ . In all cases, the skyline must have at least  $n_c + 1$  mappings.

Now assume the skyline of  $r_{\text{valid}} \vee r_{\text{mask}}$  has at least  $n_c + 1$  mappings. By construction,  $r_{\text{mask}}$  captures exactly  $n_c$  maximal mappings, so there is at least one mapping  $m$  in the skyline which is captured by  $r_{\text{valid}}$ . This mapping  $m$  encodes an assignment  $v$  of the variables of  $F$ . As  $m$  is not dominated by any mapping captured by  $r_{\text{mask}}$ , for each clause index  $j \in [n_c]$  there must exist a variable index  $i \in [n_x]$  such that  $v_{i,j}$  is assigned. Therefore  $v$  is a satisfying assignment of  $F$ . Overall, we have shown that  $F$  is satisfiable if and only if  $\eta_{\text{varInc}}(r_{\text{valid}} \vee r_{\text{mask}})$  has at least  $n_c + 1$  satisfying mappings, which concludes the proof. ◀

We last notice that we can modify Corollary 6.5 slightly: instead of applying to a fixed variable-wise domination rule (defined by fixing a single-variable domination rule), the result also applies when the domination rule is specified explicitly on the entire domain as a regular spanner:

► **Corollary 6.6.** *Assuming  $P \neq NP$ , there is no algorithm for the skyline extraction problem which is output polynomial in combined complexity even if the domination rule is given as one sequential VA (not by implicitly taking the product of single-variable sequential VAs).*

## 7 Intractable and Tractable Domination Rules

We have shown that the skyline extraction problem is intractable in combined complexity for regular spanners, and this intractability already holds in the case of a fixed variable-wise domination rule, namely, the variable inclusion rule. However, this leaves open the same question for other domination rules, e.g., for the span inclusion rule – in particular if we restrict our attention to schema-based spanners, which are typically better-behaved (e.g., for the complexity of the join and difference operators [10]).

In this section, we show that, unfortunately, hardness still holds in that context. Specifically, we introduce a condition on domination rules, called having *unboundedly many disjoint strict domination pairs* (UMDSDP). This condition is clearly satisfied by our example domination rules (except self-domination). We then show that UMDSDP is a sufficient condition for intractability: this result re-captures the hardness of variable inclusion (Theorem 6.3) and also shows hardness for the span inclusion, left-to-right, and span length domination rules.

We then introduce a restricted class of domination rules, called *variable inclusion-like*, and show that on this class a variant of the UMDSDP condition in fact *characterizes* the intractable cases. In particular, all such domination rules without the condition enjoy tractable skyline extraction. Last, we study additional examples for general domination rules, and show that among rules not covered by UMDSDP, some are easy and some are hard.

**The UMDSDP condition.** To introduce our sufficient condition for intractability of skyline extraction, we first define *disjoint strict domination pairs*.

► **Definition 7.1.** *Fixing a variable  $x$ , a single-variable mapping is a mapping  $m$  using only the variable  $x$ : the mapping  $m$  may map  $x$  to a span  $m(x)$ , or it may not map  $x$  to anything, which is represented by the special symbol “–”.*

*For a domination relation  $\leq$  on a document  $d$ , a domination pair of  $\leq$  on  $d$  is a pair  $(m_1, m_2)$  of single-variable mappings with  $m_1 \leq m_2$ . The domination pair is strict if  $m_1 \neq m_2$ .*

*Two strict domination pairs  $(m_1, m_2)$  and  $(m'_1, m'_2)$  are disjoint if, letting  $s$  be the smallest span containing the spans  $m_1(x)$  and  $m_2(x)$  if defined, and letting  $s'$  be the smallest span containing the spans  $m'_1(x)$  and  $m'_2(x)$  if defined, then  $s$  and  $s'$  are disjoint spans. Otherwise, the two strict domination pairs overlap.*

Note that, in a strict domination pair  $(m_1, m_2)$ , at least one of  $m_1$  and  $m_2$  has to assign  $x$  to a span; and if one of them does not, then the resulting unassigned span (“–”) is not taken into account. In what follows, we abuse notation and identify single-variable mappings with the span to which they map  $x$ , or identify them to “–” if they do not map  $x$  to anything.

► **Example 7.2.** The pairs  $([1, 3], [2, 4])$  and  $([9, 10], [6, 8])$  are disjoint. The pairs  $([1, 3], [7, 9])$  and  $([4, 6], [10, 12])$  overlap (even though all of the constituent spans are disjoint). Finally,  $(-, [1, 3])$  and  $([4, 6], [10, 12])$  are also disjoint.

We can now define the UMDSDP condition, which will be sufficient to show hardness:

► **Definition 7.3.** *A single-variable domination rule  $D$  has unboundedly many disjoint strict domination pairs (UMDSDP) if, given  $n \in \mathbb{N}$ , we can compute in time polynomial in  $n$  a document  $d \in \Sigma^*$  and  $n$  strict domination pairs  $S_1, \dots, S_n$  of  $D(d)$  that are pairwise disjoint.*

► **Example 7.4.**  $D_{self}$  does not satisfy UMDSDP as it has no strict domination pairs.

The span length domination rule satisfies UMDSDP. Indeed, for  $n \in \mathbb{N}$ , we can take the word  $a^n$  and the disjoint strict domination pairs  $\{([i, i], [i, i + 1]) \mid i \in [0, n - 1]\}$ . The same pairs show that UMDSDP holds for the span inclusion rule and for the left-to-right rule.

Finally, the variable inclusion domination rule satisfies UMDSDP with the set of pairs  $\{(-, [i, i + 1]) \mid i \in [0, n - 1]\}$ .

Consider the domination rule  $D_{start}$  defining the domination relation  $\leq_{start}$  that contains the pairs  $\{([1, i], [1, j]) \mid i, j \in \mathbb{N}, i \leq j\}$  plus the trivial pair  $(-, -)$  for reflexivity. Then  $\leq_{start}$  has unboundedly many strict domination pairs, but no two of them are disjoint, so the UMDSDP condition is not respected. (However, we will still be able to show intractability for this rule; see Proposition 7.9.)



We remark that, for single-variable domination rules that are regular, the UMDS DP condition holds whenever there *exist* arbitrarily many pairwise disjoint strict domination pairs (i.e., in this case we can always efficiently compute them); see the full version [4] for details.

**UMDS DP implies hardness.** We now show that the UMDS DP condition implies that skyline extraction is hard. The proof is a variant of the one for variable inclusion:

► **Theorem 7.5.** *Let  $D$  be a single-variable domination rule satisfying UMDS DP. The skyline extraction problem for  $D$ , given a sequential VA  $\mathcal{A}$  and a document  $d \in \Sigma^*$ , is not output-polynomial unless  $P = NP$ .*

This implies the hardness of the other variable-wise domination rules presented earlier, completing Corollary 6.5. Note that these rules are schema-based spanners, and we can also notice that hardness already holds if the input spanner is functional, i.e., schema-based:

► **Corollary 7.6.** *There is no algorithm for the skyline extraction problem with respect to the span inclusion domination rule or the left-to-right domination rule or the span length domination rule which is output-polynomial in combined complexity, unless  $P = NP$ . This holds even if the input VA is required to be functional.*

**Variable inclusion-like rules.** We have seen that the UMDS DP condition is a sufficient condition for skyline extraction to be hard, but this leaves open the question of whether it is necessary. We will now focus on a fragment of domination rules which we call *variable inclusion-like domination rules*, where this is the case. Formally, we say that a domination relation  $\preceq$  is *variable inclusion-like* if for all strict domination pairs  $(m_1, m_2)$  we have for all  $x \in \text{Variables}$  that if  $m_1(x)$  is defined, then  $m_2(x)$  is defined as well and  $m_1(x) = m_2(x)$ .

In contrast with the variable inclusion rule that contains all such pairs  $(m_1, m_2)$ , we only require that a subset of them hold in  $\preceq$ . We will define variable inclusion-like domination rules in a variable-wise fashion: for single-variable variable inclusion-like rules, the strict domination pairs are necessarily of the form  $(-, s)$  for a span  $s$ . In other words, a variable-wise inclusion-like domination rule is defined by indicating, on each document, which spans  $s$  can appear as the right-hand-side of such a pair. Further, for variable inclusion-like rules, two strict domination pairs are disjoint if and only if their right-hand-sides are.

We can show that, on variable inclusion-like domination rules, we have a dichotomy on a variant of the UMDS DP condition:

► **Theorem 7.7.** *Let  $D$  be a single-variable domination rule which is variable inclusion-like. If  $D$  satisfies the UMDS DP condition or accepts a pair of the form  $(-, [i, i])$  on some document, then the skyline extraction problem for  $D$ , given a sequential VA and document, is not output-polynomial in combined complexity unless  $P = NP$ . Otherwise, the skyline extraction problem for  $D$  is output-polynomial in combined complexity.*

The lower bound of the dichotomy follows from Theorem 7.5, plus the observation that a single pair of the form  $(-, [i, i])$  is sufficient to show hardness:

► **Lemma 7.8.** *Let  $D$  be a single-variable domination rule that accepts on some document a pair  $(-, [i, i])$ . Then the skyline extraction problem for  $D$ , given a sequential VA and document, is not output-polynomial in combined complexity unless  $P = NP$ .*

Hence, the interesting result in Theorem 7.7 is the upper bound. We show it in the full version [4] by observing that the set of right-hand-sides of strict domination pairs for variable inclusion-like rules that do not satisfy UMDS DP have bounded hitting set number, and showing that this implies tractability.



**Other cases.** Theorems 7.5 and 7.7 do not settle the complexity of non-UMDS DP domination rules which are not variable inclusion-like. We conclude with some examples of rules that can be shown to be intractable. We first show it for the rule  $\leq_{start}$  introduced earlier:

► **Proposition 7.9.** *Refer back to the variable-wise domination rule  $D_{start}$  from Example 7.4. There is no output-polynomial combined complexity algorithm for the skyline extraction problem for that rule, assuming  $P \neq NP$ .*

We show hardness for another rule that fails the UMDS DP, where all strict domination pairs share the same right-hand-side:

► **Proposition 7.10.** *Consider the variable-wise domination rule expressed by the regular expression  $x\{a^*\}a^*x^\dagger\{b\} \vee D_{self}$ . There is no output-polynomial combined complexity algorithm for the skyline extraction problem for that rule, assuming  $P \neq NP$ .*

We note, however, that the *reverse* of that rule, where all strict domination pairs share the same left-hand-side, is in fact tractable (and also fails the UMDS DP). This illustrates that, counter-intuitively, a complexity classification on variable-wise domination rules would not be symmetric between the left-hand-side and right-hand-side:

► **Proposition 7.11.** *The skyline extraction problem for the variable-wise domination rule  $x^\dagger\{a^*\}a^*x\{b\} \vee D_{self}$  is output-polynomial in combined complexity.*

## 8 Conclusions

We have introduced the general framework of domination rules to express the skyline operator for document spanners, with rules that are themselves expressed as a spanner. We have shown that this operator (with regular rules) does not increase the expressiveness of regular spanners, but that it incurs an unavoidable exponential blowup in the state complexity and is intractable to evaluate in combined complexity for many natural fixed rules.

Our work leaves several questions open for future investigation. The most immediate question is whether the skyline extraction problem admits a dichotomy on the variable-wise regular domination rule in the general case, i.e., extending Theorem 7.5 to arbitrary such rules. However, this seems challenging. Another question is whether the hardness results of Section 7 also give state complexity lower bounds of the kind shown in Section 5, in particular in the schema-based context; and whether there is a dichotomy on state complexity.

Last, an intriguing question is whether the *top-k problem* of computing a constant number  $k$  of mappings from the skyline is always tractable in combined complexity. None of our hardness results precludes it, but we are not aware of an algorithm for that problem.

---

## References

- 1 Foto N. Afrati, Paraschos Koutris, Dan Suciu, and Jeffrey D. Ullman. Parallel skyline queries. In *ICDT*, 2012. doi:10.1145/2274576.2274605.
- 2 Shqiponja Ahmetaj, Wolfgang Fischl, Markus Kröll, Reinhard Pichler, Mantas Šimkus, and Sebastian Skritek. The challenge of optional matching in SPARQL. In *FoIKS*, 2016. doi:10.1007/978-3-319-30024-5\_10.
- 3 Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. Constant-delay enumeration for nondeterministic document spanners. In *ICDT*, 2019. doi:10.4230/LIPIcs.ICDT.2019.22.
- 4 Antoine Amarilli, Benny Kimelfeld, Sébastien Labbé, and Stefan Mengel. Skyline operators for document spanners. *CoRR*, 2024. Full version of this article with all proofs. doi:10.48550/arXiv.2304.06155.

- 5 Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker. The skyline operator. In *ICDE*. IEEE, 2001. doi:10.1109/ICDE.2001.914855.
- 6 Craig Boutilier, Ronen I. Brafman, Carmel Domshlak, Holger H. Hoos, and David Poole. CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *J. Artif. Intell. Res.*, 21, 2004. doi:10.1613/jair.1234.
- 7 Simone Bova, Florent Capelli, Stefan Mengel, and Friedrich Slivovsky. A strongly exponential separation of DNNFs from CNF formulas. *CoRR*, abs/1411.1995, 2014. doi:10.48550/arXiv.1411.1995.
- 8 Florent Capelli and Yann Strozecki. Incremental delay enumeration: Space and time. *Discret. Appl. Math.*, 268, 2019. doi:10.1016/j.dam.2018.06.038.
- 9 Hamish Cunningham, Kevin Humphreys, Robert J. Gaizauskas, and Yorick Wilks. GATE – A general architecture for text engineering. In *ANLP*. ACL, 1997. doi:10.3115/974281.974299.
- 10 Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn Vansummeren. Document spanners: A formal approach to information extraction. *J. ACM*, 62(2):12, 2015. doi:10.1145/2699442.
- 11 Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn Vansummeren. Declarative cleaning of inconsistencies in information extraction. *ACM Trans. Database Syst.*, 41(1), 2016. doi:10.1145/2877202.
- 12 Fernando Florenzano, Cristian Riveros, Martín Ugarte, Stijn Vansummeren, and Domagoj Vrgoc. Constant delay algorithms for regular document spanners. In *PODS*, 2018. doi:10.1145/3196959.3196987.
- 13 Dominik D. Freydenberger, Benny Kimelfeld, and Liat Peterfreund. Joining extractions of regular expressions. In *PODS*, 2018. doi:10.1145/3196959.3196967.
- 14 César A. Galindo-Legaria. Outerjoins as disjunctions. *SIGMOD Rec.*, 23(2), 1994. doi:10.1145/191843.191908.
- 15 Yuan Gao, Nelma Moreira, Rogério Reis, and Sheng Yu. A survey on operational state complexity. *J. Autom. Lang. Comb.*, 21(4):251–310, 2017. doi:10.25596/jalc-2016-251.
- 16 Benoit Groz and Tova Milo. Skyline queries with noisy comparisons. In *PODS*, 2015. doi:10.1145/2745754.2745775.
- 17 Yunyao Li, Frederick Reiss, and Laura Chiticariu. SystemT: A declarative information extraction system. In *ACL*, 2011. URL: <https://aclanthology.org/P11-4019/>.
- 18 Francisco Maturana, Cristian Riveros, and Domagoj Vrgoc. Document spanners for extracting incomplete information: Expressiveness and complexity. In *PODS*, 2018. doi:10.1145/3196959.3196968.
- 19 Liat Peterfreund. *The Complexity of Relational Queries over Extractions from Text*. PhD thesis, Technion - Computer Science Department, 2019. URL: <https://www.cs.technion.ac.il/users/wwwb/cgi-bin/tr-info.cgi/2019/PHD/PHD-2019-10>.
- 20 Liat Peterfreund. Grammars for document spanners. In *ICDT*, 2021. doi:10.4230/LIPICS.ICDT.2021.7.
- 21 Liat Peterfreund, Dominik D. Freydenberger, Benny Kimelfeld, and Markus Kröll. Complexity bounds for relational algebra over document spanners. In *PODS*, 2019. doi:10.1145/3294052.3319699.
- 22 Liat Peterfreund, Balder ten Cate, Ronald Fagin, and Benny Kimelfeld. Recursive programs for document spanners. In *ICDT*, 2019. doi:10.4230/LIPICS.ICDT.2019.13.
- 23 Igor Razgon. On the read-once property of branching programs and CNFs of bounded treewidth. *Algorithmica*, 75(2), 2016. doi:10.1007/s00453-015-0059-x.
- 24 Markus L. Schmid and Nicole Schweikardt. A purely regular approach to non-regular core spanners. In *ICDT*, 2021. doi:10.4230/LIPICS.ICDT.2021.4.
- 25 Cheng Sheng and Yufei Tao. Worst-case I/O-efficient skyline algorithms. *ACM Transactions on Database Systems (TODS)*, 37(4), 2012. doi:10.1145/2389241.2389245.
- 26 Slawek Staworko, Jan Chomicki, and Jerzy Marcinkowski. Prioritized repairing and consistent query answering in relational databases. *Ann. Math. Artif. Intell.*, 64(2-3), 2012. doi:10.1007/s10472-012-9288-8.

## A Spanner Algebra

In this appendix, we introduce some operators on spanners that are used in the main text.

For every spanner  $P$  and every subset  $Y \subseteq \text{SVars}(P)$ , we define the projection operator  $\pi_Y$  by saying that  $\pi_Y P$  is the spanner that extracts on every document  $d$  the set  $(\pi_Y P)(d) := \{m|_Y \mid m \in P(d)\}$  where  $m|_Y$  is the restriction of  $m$  to  $Y$ .

The *natural join*  $P_1 \bowtie P_2$  of two spanners  $P_1$  and  $P_2$  is a spanner which accepts all the mappings  $m$  which are the union of two compatible mappings  $m_1$  accepted by  $P_1$  and  $m_2$  accepted by  $P_2$ . Said differently,  $(P_1 \bowtie P_2)(d) := \{m \in \text{Maps} \mid \exists m_1 \in P_1(d), \exists m_2 \in P_2(d), m_1 \sim m \wedge m_2 \sim m \wedge \text{dom}(m_1) \cup \text{dom}(m_2) = \text{dom}(m)\}$ .

We remark that if  $\text{SVars}(P_1) \cap \text{SVars}(P_2) = \emptyset$  then the join operator is the Cartesian product defined before.

The intersection operator  $\cap$  is defined to compute the spanner  $P_1 \cap P_2$  which on every document computes the set  $(P_1 \cap P_2)(d) := P_1(d) \cap P_2(d)$ . Observe that that if  $\text{SVars}(P_1) = \text{SVars}(P_2)$  and both spanners are schema-based, then the join operator is the intersection:  $(P_1 \bowtie P_2)(d) = P_1(d) \cap P_2(d)$ .

The union  $P_1 \cup P_2$  is defined to as the spanner which on every document computes the set  $(P_1 \cup P_2)(d) := P_1(d) \cup P_2(d)$ .

The *difference*,  $P_1 - P_2$  is a binary operator which accepts all mappings accepted by  $P_1$  which are not accepted by  $P_2$ . Said differently,  $(P_1 - P_2)(d) := P_1(d) \setminus P_2(d)$ . Note that this is the usual difference operator on sets, and *not* the difference operator defined in [19] which accepts mappings of  $P_1$  for which no compatible mapping is accepted by  $P_2$ .

It is known that the projection, natural join operator and union operators do not increase the expressive power of regular spanners, see [10] for the case of schema-based spanners and [18] for schemaless spanners. It follows that the same is true for the Cartesian product operator. We show in the full version [4] that intersection does not increase the expressivity, either. As for the difference operator, the same result is proven in [10] for schema-based regular spanners, but we are not aware of the same result for schemaless spanners and for our semantics of difference, so we prove it in the full version [4].



# When Do Homomorphism Counts Help in Query Algorithms?

Balder ten Cate  

University of Amsterdam, The Netherlands



Victor Dalmau  

Universitat Pompeu Fabra, Barcelona, Spain

Phokion G. Kolaitis  

University of California Santa Cruz, CA, USA

IBM Almaden Research Center, San Jose, CA, USA

Wei-Lin Wu  

University of California Santa Cruz, CA, USA

---

## Abstract

A query algorithm based on homomorphism counts is a procedure for determining whether a given instance satisfies a property by counting homomorphisms between the given instance and finitely many predetermined instances. In a left query algorithm, we count homomorphisms from the predetermined instances to the given instance, while in a right query algorithm we count homomorphisms from the given instance to the predetermined instances. Homomorphisms are usually counted over the semiring  $\mathbb{N}$  of non-negative integers; it is also meaningful, however, to count homomorphisms over the Boolean semiring  $\mathbb{B}$ , in which case the homomorphism count indicates whether or not a homomorphism exists. We first characterize the properties that admit a left query algorithm over  $\mathbb{B}$  by showing that these are precisely the properties that are both first-order definable and closed under homomorphic equivalence. After this, we turn attention to a comparison between left query algorithms over  $\mathbb{B}$  and left query algorithms over  $\mathbb{N}$ . In general, there are properties that admit a left query algorithm over  $\mathbb{N}$  but not over  $\mathbb{B}$ . The main result of this paper asserts that if a property is closed under homomorphic equivalence, then that property admits a left query algorithm over  $\mathbb{B}$  if and only if it admits a left query algorithm over  $\mathbb{N}$ . In other words and rather surprisingly, homomorphism counts over  $\mathbb{N}$  do *not* help as regards properties that are closed under homomorphic equivalence. Finally, we characterize the properties that admit both a left query algorithm over  $\mathbb{B}$  and a right query algorithm over  $\mathbb{B}$ .

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Logic and databases

**Keywords and phrases** query algorithms, homomorphism, homomorphism counts, conjunctive query, constraint satisfaction

**Digital Object Identifier** 10.4230/LIPIcs.ICDT.2024.8

**Funding** *Balder ten Cate*: Supported by the European Union’s Horizon 2020 research and innovation programme (MSCA-101031081).

*Victor Dalmau*: Supported by the MiCin under grants PID2019-109137GB-C22 and PID2022-138506NB-C22, and the Maria de Maeztu program (CEX2021-001195-M).

*Phokion G. Kolaitis*: Partially supported by NSF Grant IIS-1814152.

## 1 Introduction

Consider a scenario in which we are interested in a certain property of database instances and we wish to find out whether or not a given instance  $A$  satisfies that property by asking finitely many predetermined queries against  $A$ . Naturally, which properties can be checked in this way depends on what kind of queries we are allowed to ask. For example, if we are restricted to using Boolean conjunctive queries and evaluating them under set semantics,



© Balder ten Cate, Victor Dalmau, Phokion G. Kolaitis, and Wei-Lin Wu; licensed under Creative Commons License CC-BY 4.0

27th International Conference on Database Theory (ICDT 2024).

Editors: Graham Cormode and Michael Shekelyan; Article No. 8; pp. 8:1–8:20

Leibniz International Proceedings in Informatics



Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

then only properties that are invariant under homomorphic equivalence stand a chance to be checked in this way. In particular, this means that we cannot test in this way whether a relation in a given instance contains precisely 5 tuples. In contrast, if we are also allowed to ask for the number of homomorphisms from a given conjunctive query to  $A$  (a feature that is supported by actual database management systems), then we can find out much more about the instance  $A$ , including whether it satisfies the aforementioned property.

In this paper, we embark on a systematic study of this scenario. As usual, by a *property* of instances we mean a class  $\mathcal{C}$  of instances closed under isomorphisms. We write  $\mathbb{B}$  for the Boolean semiring with  $\vee$  and  $\wedge$  as its operations, while we write  $\mathbb{N}$  for the semiring of the non-negative integers with  $+$  and  $\times$  as its operations. Formally, we say that a class  $\mathcal{C}$  of instances admits a *left query algorithm over*  $\mathbb{B}$  if there exists a finite set  $\mathcal{F} = \{F_1, \dots, F_k\}$  of instances such that the membership in  $\mathcal{C}$  of an arbitrary instance  $A$  is completely determined by  $\text{hom}_{\mathbb{B}}(\mathcal{F}, A)$ , where  $\text{hom}_{\mathbb{B}}(\mathcal{F}, A)$  is the  $k$ -tuple of Boolean values indicating for each  $i \leq k$  whether or not there is a homomorphism  $F_i \rightarrow A$ . Similarly, we say that a class  $\mathcal{C}$  of instances admits a *left query algorithm over*  $\mathbb{N}$  if there exists a finite set  $\mathcal{F} = \{F_1, \dots, F_k\}$  of instances such that the membership in  $\mathcal{C}$  of an arbitrary instance  $A$  is completely determined by  $\text{hom}_{\mathbb{N}}(\mathcal{F}, A)$ , where  $\text{hom}_{\mathbb{N}}(\mathcal{F}, A)$  is the  $k$ -tuple of non-negative integers indicating for each  $i \leq k$  how many homomorphisms from  $F_i$  to  $A$  there are. *Right query algorithms* over  $\mathbb{B}$  and over  $\mathbb{N}$  are defined similarly, except that we now count the homomorphisms from  $A$  to each  $F_i$  instead of the homomorphisms from each  $F_i$  to  $A$ .

Assume that the class  $\mathcal{C}$  of instances under consideration admits a left query algorithm over  $\mathbb{B}$  or over  $\mathbb{N}$  using the set  $\mathcal{F} = \{F_1, \dots, F_k\}$ . Let  $q^{F_i}$  be the canonical conjunctive query associated with the instance  $F_i$ ,  $1 \leq i \leq k$ . Then  $\text{hom}_{\mathbb{B}}(\mathcal{F}, A)$  is the  $k$ -tuple of Boolean values denoting whether  $q^{F_i}(A) = 1$  or  $q^{F_i}(A) = 0$ , i.e., whether  $q^{F_i}$  is true or false on  $A$  under set semantics. Similarly,  $\text{hom}_{\mathbb{N}}(\mathcal{F}, A)$  is the  $k$ -tuple of non-negative integers that are the answers to  $q^{F_i}$  on  $A$  under bag-set semantics. Thus, intuitively, a class  $\mathcal{C}$  admits a left query algorithm over  $\mathbb{B}$  or over  $\mathbb{N}$  if membership in  $\mathcal{C}$  is answerable by evaluating finitely many Boolean conjunctive queries under set semantics or under bag-set semantics, respectively. Observe that these are data complexity notions because the queries  $q^{F_1}, \dots, q^{F_k}$  are fixed while the instance  $A$  varies. Observe also that these two notions differ in expressive power: for example, if  $\mathcal{C}$  is the class of instances in which a particular relation  $R$  has precisely 5 tuples, then  $\mathcal{C}$  admits a left query algorithm over  $\mathbb{N}$ , but not over  $\mathbb{B}$ .

There are two pieces of earlier work (each with different motivation and results) where the notion of a left query algorithm or variants of this notion have been explored. First, Bielecki and Van den Bussche [4] defined what it means for a query  $p$  to be *derivable* through interrogation with a query language  $L$  using a *database independent strategy*, where the interrogation consists of asking the cardinality  $|q(A)|$  for finitely many queries  $q \in L$ . When  $L$  is the language of conjunctive queries with no existential quantifiers, such strategies correspond to left-query algorithms over  $\mathbb{N}$ ; whereas, when  $L$  is the language of Boolean conjunctive queries, they correspond to left-query algorithms over  $\mathbb{B}$ . Second, when the instances are unordered graphs, the concept of a left query algorithm over  $\mathbb{N}$  was studied by Chen et al. [9] under the name *non-adaptive query algorithm*; note that Chen et al. [9] were apparently unaware of the work by Bielecki and Van den Bussche [4]. The term “non-adaptive” is apt as it conveys that the instances  $F_i$  (or the associated conjunctive queries  $q^{F_i}$ ),  $1 \leq i \leq k$ , in the set  $\mathcal{F}$  depend only on the class  $\mathcal{C}$  and do not change during a run of the query algorithm. It is also natural to consider *adaptive query algorithms*, where the instances  $F_i$ ,  $1 \leq i \leq k$ , are not required to be fixed up front. In fact, such adaptive notions were explored in both [4] and [9]. In particular, Chen et al. [9] showed that adaptive left

query algorithms over  $\mathbb{N}$  are more powerful than non-adaptive ones over  $\mathbb{N}$ . It is easy to see, however, that the existence of an adaptive left query algorithm over  $\mathbb{B}$  implies the existence of a non-adaptive one over  $\mathbb{B}$ . In this sense, adaptive left query algorithms over  $\mathbb{B}$  are not more powerful than non-adaptive ones over  $\mathbb{B}$ . Also, since in this paper we study non-adaptive algorithms only (but over both  $\mathbb{B}$  and  $\mathbb{N}$ ), we will not use the adjective “non-adaptive” here.

Our investigation begins by focusing on left query algorithms over  $\mathbb{B}$ . It is easy to see that a class of instances admits a left query algorithm over  $\mathbb{B}$  if and only if it is definable by a Boolean combination of conjunctive queries. Using tools developed by Rossman [27] to prove the preservation-under-homomorphisms theorem in the finite, we obtain a deeper characterization of such classes by showing that a class of instances admits a left query algorithm over  $\mathbb{B}$  if and only if it is both first-order definable and closed under homomorphic equivalence. Clearly, if a class of instances is closed under homomorphic equivalence, then it is a (possibly infinite) union of homomorphic equivalence classes. We show that if  $\mathcal{C}$  is a finite union of homomorphic-equivalence classes, then  $\mathcal{C}$  admits a left query algorithm over  $\mathbb{B}$  if and only if every homomorphic-equivalence class in that union admits a left query algorithm over  $\mathbb{B}$ . In contrast, a similar result does not hold for arbitrary infinite unions.

After this, we turn attention to a comparison between left query algorithms over  $\mathbb{B}$  and left query algorithms over  $\mathbb{N}$ . As discussed earlier, left query algorithms over  $\mathbb{N}$  are more powerful than left query algorithms over  $\mathbb{B}$ . The intuitive reason is that left query algorithms over  $\mathbb{B}$  do not differentiate between homomorphically equivalent instances, while those over  $\mathbb{N}$  do. The main (and technically more challenging) result of this paper reveals that, in a precise sense, this is the *only* reason why left query algorithms over  $\mathbb{N}$  are more powerful than left query algorithms over  $\mathbb{B}$ . More precisely, our main theorem asserts that if a class  $\mathcal{C}$  is closed under homomorphic equivalence, then  $\mathcal{C}$  admits a left query algorithm over  $\mathbb{B}$  if and only if  $\mathcal{C}$  admits a left query algorithm over  $\mathbb{N}$ . In other words and rather surprisingly, homomorphism counts over  $\mathbb{N}$  do *not* help as regards properties that are closed under homomorphic equivalence. As an immediate consequence, a constraint satisfaction problem has a left query algorithm over  $\mathbb{N}$  if and only if this problem is first-order definable.

Finally, we characterize the properties that admit both a left query algorithm over  $\mathbb{B}$  and a right query algorithm over  $\mathbb{B}$ . In particular, we show that a class  $\mathcal{C}$  of instances admits both a left query algorithm over  $\mathbb{B}$  and a right query algorithm over  $\mathbb{B}$  if and only if  $\mathcal{C}$  is definable by a Boolean combination of Berge-acyclic conjunctive queries. To see the point of this result, recall that if a class admits a left query algorithm over  $\mathbb{B}$ , then it is definable by a Boolean combination of conjunctive queries. Thus, if the class admits also a right query algorithm over  $\mathbb{B}$ , then these conjunctive queries can be taken to be Berge-acyclic.

**Related work.** A classical result by Lovász [25] characterizes graph isomorphism in terms of “left” homomorphism counts: two graphs  $G$  and  $H$  are isomorphic if and only if for every graph  $F$ , the number of homomorphisms from  $F$  to  $G$  is equal to the number of homomorphisms from  $F$  to  $H$ . In more recent years, there has been a study of relaxations of isomorphisms obtained by requiring that the number of homomorphisms from  $F$  to  $G$  is equal to the number of homomorphisms from  $F$  to  $H$ , where  $F$  ranges over a restricted class of graphs [12, 11, 6]. Furthermore, a study of relaxations of isomorphism obtained by counting the number of “right” homomorphisms to a restricted class was carried out in [3].

There has been an extensive body of research on answering queries under various types of access restrictions. Closer in spirit to the work reported here is view determinacy, which is the question of whether the answers to a query can be inferred when given access only to a certain view of the database instance [26]. We note that view determinacy is typically

concerned with non-Boolean queries and non-Boolean views; in contrast, the question of whether a class admits a left query algorithm over  $\mathbb{B}$  can be interpreted as the question of whether there is a finite set of Boolean conjunctive queries that determine a given Boolean query. A study of view determinacy under bag-set semantics was recently initiated in [24]. Section 7 contains additional commentary on the relationship between left query algorithms over  $\mathbb{N}$  and view determinacy under bag-set semantics.

## 2 Basic Notions

**Relational database instances.** A *relational database schema* or, simply, a *schema* is a finite set  $\sigma = \{R_1, \dots, R_m\}$  of relation symbols  $R_i$ , each of which has a positive integer  $r_i$  as its associated *arity*. A *relational database instance* or, simply, an *instance* is a tuple  $A = (R_1^A, \dots, R_m^A)$ , where each  $R_i^A$  is a relation of arity  $r_i$ . The *facts* of the instance  $A$  are the tuples in the relations  $R_i^A$ ,  $1 \leq i \leq m$ . The *active domain* of  $A$ , denoted  $\text{adom}(A)$ , is the set of all entries occurring in the facts of  $A$ . All instances  $A$  considered are assumed to be finite, i.e.,  $\text{adom}(A)$  is finite. A *graph* is an instance  $A$  over a schema consisting of a binary relation symbol  $E$  and such that  $E^A$  is a binary relation that is symmetric and irreflexive.

The *incidence multigraph*  $\text{inc}(A)$  of an instance  $A$  is the bipartite multigraph whose parts are the sets  $\text{adom}(A)$  and  $\text{block}(A) = \{(R, t) : R \in \sigma \text{ and } t \in R^A\}$ , and whose edges are the pairs  $(a, (R, t))$  such that  $a$  is one of the entries of  $t$ . A *path of length  $n$*  in  $A$  is a sequence  $a_0, a_1, \dots, a_n$  of elements in  $\text{adom}(A)$  for which there are elements  $b_1, \dots, b_n$  in  $\text{block}(A)$  such that the sequence  $a_0, b_1, a_1, \dots, b_n, a_n$  is a path in  $\text{inc}(A)$  in the standard graph-theoretic sense (disallowing traversing an edge twice in succession in opposite directions). Two elements  $a$  and  $a'$  in  $\text{adom}(A)$  are *connected* if  $a = a'$  or there is a path  $a_0, a_1, \dots, a_n$  in  $A$  with  $a = a_0$  and  $a' = a_n$ . We say that  $A$  is *connected* if every two elements  $a$  and  $a'$  in  $\text{adom}(A)$  are connected. A *cycle of length  $n$*  in  $A$  is a path of length  $n$  in  $A$  with  $a_n = a_0$ . We say  $A$  is *acyclic* if it contains no cycles. The *girth* of  $A$  is the shortest length of a cycle in  $A$  or  $\infty$  if  $A$  is acyclic.

An instance  $A$  is a *subinstance* of an instance  $B$  if  $R^A \subseteq R^B$ , for every  $R \in \sigma$ .

A *class*  $\mathcal{C}$  of instances is a collection of instances over the same schema that is closed under isomorphism (i.e., if  $A \in \mathcal{C}$  and  $B$  is isomorphic to  $A$ , then  $B \in \mathcal{C}$ ). Every decision problem  $P$  about instances can be identified with the class of all “yes” instances of  $P$ .

**Homomorphisms, conjunctive queries, and canonical instances.** A *homomorphism* from an instance  $A$  to an instance  $B$  is a function  $h : \text{adom}(A) \rightarrow \text{adom}(B)$  such that for every relation symbol  $R \in \sigma$  with arity  $r$  and for all elements  $a_1, \dots, a_r$  in  $\text{adom}(A)$  with  $(a_1, \dots, a_r) \in R^A$ , we have  $(h(a_1), \dots, h(a_r)) \in R^B$ . We write  $h : A \rightarrow B$  to denote that  $h$  is a homomorphism from  $A$  to  $B$ ; we also write  $A \rightarrow B$  to denote that there is a homomorphism from  $A$  to  $B$ . We say that  $A$  and  $B$  are *homomorphically equivalent*, denoted  $A \leftrightarrow B$ , if  $A \rightarrow B$  and  $B \rightarrow A$ . Clearly,  $\leftrightarrow$  is an equivalence relation on instances. We write  $[A]_{\leftrightarrow}$  to denote the equivalence class of  $A$  with respect to  $\leftrightarrow$ , i.e.,  $[A]_{\leftrightarrow} = \{B : B \leftrightarrow A\}$ .

Let  $\mathcal{C}$  be a class of instances. We say that  $\mathcal{C}$  is *closed under homomorphic equivalence* if whenever  $A \in \mathcal{C}$  and  $A \leftrightarrow B$ , we have that  $B \in \mathcal{C}$ . As an example, for every instance  $A$ , we have that the equivalence class  $[A]_{\leftrightarrow}$  is closed under homomorphic equivalence. For a different example, the class of all 3-colorable graphs is closed under homomorphic equivalence.

We assume familiarity with the syntax and the semantics of first-order logic (FO). For a FO-sentence  $\varphi$ , we denote by  $\text{Mod}(\varphi)$  the set  $\{A : A \models \varphi\}$  of instances  $A$  that satisfy  $\varphi$  under the active domain semantics (i.e., the quantifiers range over elements of the active domain of



the instance at hand). A *Boolean conjunctive query* (Boolean CQ) is a FO-sentence of the form  $\exists x_1 \dots x_n (\alpha_1 \wedge \dots \wedge \alpha_k)$ , where each  $\alpha_j$  is an atomic formula of the form  $R(y_1, \dots, y_r)$ , each variable  $y_i$  is among the variables  $x_1, \dots, x_n$ , and each variable  $x_i$  occurs in at least one of the atomic formulas  $\alpha_1, \dots, \alpha_k$ .

The *canonical instance* of a conjunctive query  $q$ , denoted  $A^q$ , is the instance whose active domain consists of the variables of  $q$ , and whose facts are the conjuncts of  $q$ . Conversely, the *canonical conjunctive query* of an instance  $A$ , denoted  $q^A$ , has, for each  $a$  in  $\text{adom}(A)$ , an existentially quantified variable  $x_a$  and, for each fact  $(a_1, \dots, a_r) \in R^A$ , a conjunct  $R(x_{a_1}, \dots, x_{a_r})$ . An immediate consequence of the semantics of FO is that, for every two instances  $A$  and  $D$ , we have that  $D \models q^A$  if and only if  $A \rightarrow D$ .

A conjunctive query is *Berge-acyclic* if its canonical instance is acyclic, as defined earlier. The notion of Berge-acyclicity is stronger than the more standard notion of acyclicity in database theory, which requires that the conjunctive query has a join tree (see, e.g., [1]).

**Homomorphism counts, left and right profiles.** Let  $\mathbb{N} = (\mathbb{N}, +, \times, 0, 1)$  be the semiring of the non-negative integers and let  $\mathbb{B} = (\{0, 1\}, \vee, \wedge, 0, 1)$  be the Boolean semiring. If  $A$  and  $B$  are two instances, then we write  $\text{hom}_{\mathbb{N}}(A, B)$  for the number of homomorphisms from  $A$  to  $B$ . For example, if  $A$  is a graph and  $K_3$  is the 3-clique, then  $\text{hom}_{\mathbb{N}}(A, K_3)$  is the number of 3-colorings of  $A$ . We extend this notion to  $\text{hom}_{\mathbb{B}}(A, B)$ , where  $\text{hom}_{\mathbb{B}}(A, B) = 1$  if there is a homomorphism from  $A$  to  $B$ , and  $\text{hom}_{\mathbb{B}}(A, B) = 0$  otherwise. For example,  $\text{hom}_{\mathbb{B}}(A, K_3) = 1$  if  $A$  is 3-colorable, and  $\text{hom}_{\mathbb{B}}(A, K_3) = 0$  if  $A$  is not 3-colorable.

Let  $\mathcal{F} = \{F_1, \dots, F_k\}$  be a finite non-empty set of instances and let  $A$  be an instance.

- The *left profile* of  $A$  in  $\mathcal{F}$  over  $\mathbb{N}$  is the tuple

$$\text{hom}_{\mathbb{N}}(\mathcal{F}, A) = (\text{hom}_{\mathbb{N}}(F_1, A), \dots, \text{hom}_{\mathbb{N}}(F_k, A)).$$

- The *left profile* of  $A$  in  $\mathcal{F}$  over  $\mathbb{B}$  is the tuple

$$\text{hom}_{\mathbb{B}}(\mathcal{F}, A) = (\text{hom}_{\mathbb{B}}(F_1, A), \dots, \text{hom}_{\mathbb{B}}(F_k, A)).$$

- The *right profile* of  $A$  in  $\mathcal{F}$  over  $\mathbb{N}$  is the tuple

$$\text{hom}_{\mathbb{N}}(A, \mathcal{F}) = (\text{hom}_{\mathbb{N}}(A, F_1), \dots, \text{hom}_{\mathbb{N}}(A, F_k)).$$

- The *right profile* of  $A$  in  $\mathcal{F}$  over  $\mathbb{B}$  is the tuple

$$\text{hom}_{\mathbb{B}}(A, \mathcal{F}) = (\text{hom}_{\mathbb{B}}(A, F_1), \dots, \text{hom}_{\mathbb{B}}(A, F_k)).$$

Let  $A_1, \dots, A_n$  be instances whose active domains are pairwise disjoint.

- The *direct sum*  $A_1 \oplus \dots \oplus A_n$  of  $A_1, \dots, A_n$  is the instance such that  $R^{A_1 \oplus \dots \oplus A_n} = R^{A_1} \cup \dots \cup R^{A_n}$ , for every  $R \in \sigma$ .
- The *direct product*  $A_1 \otimes \dots \otimes A_n$  of  $A_1, \dots, A_n$  is the instance such that the relation  $R^{A_1 \otimes \dots \otimes A_n}$  consists of all tuples  $(\mathbf{a}_1, \dots, \mathbf{a}_r)$  with  $(\mathbf{a}_1(i), \dots, \mathbf{a}_r(i)) \in R^{A_i}$ , for  $1 \leq i \leq n$  and for every  $R \in \sigma$  of arity  $r$ .

The next proposition is well known and has a straightforward proof.

► **Proposition 2.1.** *Let  $A, B_1, B_2$  be instances, and let  $K \in \{\mathbb{B}, \mathbb{N}\}$ . Then the following statements are true.*

1.  $\text{hom}_K(A, B_1 \oplus B_2) = \text{hom}_K(A, B_1) +_K \text{hom}_K(A, B_2)$ , provided that  $A$  is connected;
  2.  $\text{hom}_K(A, B_1 \otimes B_2) = \text{hom}_K(A, B_1) \cdot_K \text{hom}_K(A, B_2)$ ;
  3.  $\text{hom}_K(B_1 \oplus B_2, A) = \text{hom}_K(B_1, A) \cdot_K \text{hom}_K(B_2, A)$ ,
- where  $+_{\mathbb{N}}$  and  $\cdot_{\mathbb{N}}$  stand for addition  $+$  and multiplication  $\times$  of non-negative integers, while  $+_{\mathbb{B}}$  and  $\cdot_{\mathbb{B}}$  stand for disjunction  $\vee$  and conjunction  $\wedge$  of the Boolean values 0 and 1.

### 3 Left Query Algorithms and Right Query Algorithms

In [9], Chen et al. focused on classes of graphs and introduced the notions of a left query algorithm and a right query algorithm over the semiring  $\mathbb{N}$  of the non-negative integers. Here, we extend their framework in two ways: first, we consider classes of instances over some arbitrary, but fixed, schema; second, we consider left query algorithms and right query algorithms over the Boolean semiring  $\mathbb{B}$ .

- **Definition 3.1.** *Let  $\mathcal{C}$  be a class of instances and let  $K$  be the semiring  $\mathbb{B}$  or  $\mathbb{N}$ .*
- *Assume that  $k$  is a positive integer.*
    - *A left  $k$ -query algorithm over  $K$  for  $\mathcal{C}$  is a pair  $(\mathcal{F}, X)$ , where  $\mathcal{F} = \{F_1, \dots, F_k\}$  is a set of instances and  $X$  is a set of  $k$ -tuples over  $K$ , such that for all instances  $D$ , we have that  $D \in \mathcal{C}$  if and only if  $\text{hom}_K(\mathcal{F}, D) \in X$ .*
    - *A right  $k$ -query algorithm over  $K$  for  $\mathcal{C}$  is a pair  $(\mathcal{F}, X)$ , where  $\mathcal{F} = \{F_1, \dots, F_k\}$  is a set of instances and  $X$  is a set of  $k$ -tuples over  $K$ , such that for all instances  $D$ , we have that  $D \in \mathcal{C}$  if and only if  $\text{hom}_K(D, \mathcal{F}) \in X$ .*
  - *We say that  $\mathcal{C}$  admits a left query algorithm over  $K$  if for some  $k > 0$ , there is a left  $k$ -query algorithm over  $K$  for  $\mathcal{C}$ . Similarly, we say that  $\mathcal{C}$  admits a right query algorithm over  $K$  if for some  $k > 0$ , there is a right  $k$ -query algorithm over  $K$  for  $\mathcal{C}$ .*

The term “query algorithm” is natural because we can think of a query algorithm as a procedure for determining if a given instance belongs to the class  $\mathcal{C}$ : we compute the left homomorphism-count vector (in the case of a left query algorithm) or the right homomorphism-count vector (in the case of a right query algorithm) and test whether it belongs to  $X$ . When the semiring  $\mathbb{N}$  is considered, this is a somewhat abstract notion of an algorithm because it makes no requirements on the effectiveness of the set  $X$ . Not requiring  $X$  to be a decidable set makes our results regarding the non-existence of left query algorithms over  $\mathbb{N}$  stronger. Moreover, in all cases where we establish the existence of a left query algorithm over  $\mathbb{N}$  or a right query algorithm over  $\mathbb{N}$ , the set  $X$  will happen to be decidable (for the semiring  $\mathbb{B}$ , the set  $X$  is always finite, hence decidable).

Let  $K$  be the semiring  $\mathbb{B}$  or  $\mathbb{N}$ . It is clear that if two classes of instances admit a left query algorithm over  $K$ , then so do their complements, their union, and their intersection. Consequently, the classes of instances that admit a left query algorithm over  $K$  are closed under Boolean combinations. Furthermore, the same holds true for right query algorithms.

By Part 3 of Proposition 2.1, we have that if  $K$  is the semiring  $\mathbb{B}$  or the semiring  $\mathbb{N}$ , then  $\text{hom}_K$  is *multiplicative on the left*, i.e., for all instances  $A_1, \dots, A_n$  and  $B$ , we have  $\text{hom}_K(A_1 \oplus \dots \oplus A_n, B) = \text{hom}_K(A_1, B) \cdot_K \dots \cdot_K \text{hom}_K(A_n, B)$ . It follows that, as regards the existence of left query algorithms, we may assume that all instances in the finite set  $\mathcal{F}$  of a left query algorithm over  $\mathbb{B}$  or over  $\mathbb{N}$  are connected. We state this observation as a proposition that will be used repeatedly in the sequel.

► **Proposition 3.2.** *Let  $\mathcal{C}$  be a class of instances and let  $K$  be the semiring  $\mathbb{B}$  or  $\mathbb{N}$ . Then the following statements are equivalent.*

1.  *$\mathcal{C}$  admits a left query algorithm  $(\mathcal{F}, X)$  over  $K$ .*
2.  *$\mathcal{C}$  admits a left query algorithm  $(\mathcal{F}, X)$  over  $K$ , where every instance in the set  $\mathcal{F}$  is connected.*

Left profiles over  $\mathbb{N}$  contain more information than left profiles over  $\mathbb{B}$ . Therefore, if membership in a class  $\mathcal{C}$  can be determined using left profiles over  $\mathbb{B}$ , then it ought to be also determined using left profiles over  $\mathbb{N}$ . Similar considerations hold for right profiles. The next proposition makes these assertions precise.

► **Proposition 3.3.** *Let  $\mathcal{C}$  be a class of instances and let  $\mathcal{F}$  be a finite set of instances.*

*If  $\mathcal{C}$  admits a left query algorithm over  $\mathbb{B}$  of the form  $(\mathcal{F}, X)$  for some set  $X$ , then  $\mathcal{C}$  admits a left query algorithm over  $\mathbb{N}$  of the form  $(\mathcal{F}, X')$  for some set  $X'$ . In particular, if  $\mathcal{C}$  admits a left query algorithm over  $\mathbb{B}$ , then it also admits a left query algorithm over  $\mathbb{N}$ .*

*Furthermore, the same holds true for right query algorithms.*

**Proof.** Assume that  $\mathcal{C}$  admits a left query algorithm  $(\mathcal{F}, X)$  over  $\mathbb{B}$ , where  $\mathcal{F} = \{F_1, \dots, F_k\}$  and  $X \subseteq \{0, 1\}^k$ , for some  $k > 0$ . For every  $t = (t_1, \dots, t_k) \in \{0, 1\}^k$ , we let  $X_t$  be the set

$$X_t = \{(s_1, \dots, s_k) \in \mathbb{N}^k : s_i = 0 \text{ if and only if } t_i = 0, \text{ for } 1 \leq i \leq k\}.$$

Consider the set  $X' = \bigcup_{t \in X} X_t$ . It is easy to verify that the pair  $(\mathcal{F}, X')$  is a left  $k$ -query algorithm for  $\mathcal{C}$  over  $\mathbb{N}$ . The argument for right query algorithms is entirely analogous. ◀

As pointed out in the Introduction, the converse of Proposition 3.3 is not true, in general.

We now give several examples illustrating left and right query algorithms.

► **Example 3.4.** Let  $\mathcal{C}$  be the class of all triangle-free graphs, i.e., the graphs  $G$  for which there is no homomorphism from  $K_3$  to  $G$ . Clearly,  $\mathcal{C}$  admits a left 1-query algorithm  $(\mathcal{F}, X)$  over  $\mathbb{B}$ , where  $\mathcal{F} = \{K_3\}$  and  $X = \{0\}$  (recall that  $K_3$  is the 3-clique).<sup>1</sup> Therefore, by Proposition 3.3,  $\mathcal{C}$  admits a left 1-query algorithm over  $\mathbb{N}$ . In contrast, Chen et al. [9, Proposition 8.2] showed that (the complement of)  $\mathcal{C}$  does not admit a right query algorithm over  $\mathbb{N}$ , hence (again by Proposition 3.3) it does not admit a right query algorithm over  $\mathbb{B}$ .

We now recall the definition of constraint satisfaction problems.

► **Definition 3.5.** *If  $B$  is an instance, then the constraint satisfaction problem  $\text{CSP}(B)$  is the following decision problem: given an instance  $A$ , is there a homomorphism from  $A$  to  $B$ ?*

For  $k \geq 2$ , let  $K_k$  denote the  $k$ -clique. Then  $\text{CSP}(K_k)$  is the  $k$ -COLORABILITY problem: given a graph  $G$ , is  $G$   $k$ -colorable? During the past three decades, there has been an extensive study of complexity of constraint satisfaction problems, motivated by the Feder-Vardi Conjecture that for every instance  $B$ , either  $\text{CSP}(B)$  is NP-complete or  $\text{CSP}(B)$  is solvable in polynomial time. This conjecture was eventually confirmed independently by Bulatov [7] and Zhuk [29].

Every constraint satisfaction problem will be identified with the class of its “yes” instances, that is, for every instance  $B$ , we have that  $\text{CSP}(B) = \{A : A \rightarrow B\}$ .

► **Example 3.6.** Let  $B$  be an instance. Clearly,  $\text{CSP}(B)$  admits a right 1-query algorithm  $(\mathcal{F}, X)$  over  $\mathbb{B}$ , where  $\mathcal{F} = \{B\}$  and  $X = \{1\}$ . Therefore, by Proposition 3.3,  $\text{CSP}(B)$  admits a right 1-query algorithm  $(\mathcal{F}, X')$  over  $\mathbb{N}$ . In particular, the 3-COLORABILITY problem  $\text{CSP}(K_3)$  admits a right query algorithm over both  $\mathbb{B}$  and  $\mathbb{N}$ . In contrast, it will follow from results in Section 4 and Section 5 that  $\text{CSP}(K_3)$  does not admit a left query algorithm over  $\mathbb{B}$  or over  $\mathbb{N}$  (see Remark 4.5).

► **Example 3.7.** Consider the homomorphic equivalence class  $[K_3]_{\leftrightarrow}$ . Note that  $[K_3]_{\leftrightarrow}$  is the class of all graphs that are 3-colorable and also contain a triangle. From results in Section 4, it will follow that  $[K_3]_{\leftrightarrow}$  does not admit a left query algorithm over  $\mathbb{B}$  (see Remark 4.5). Furthermore, from results in Section 6, it will follow that  $[K_3]_{\leftrightarrow}$  does not admit a right query algorithm over  $\mathbb{B}$  (see Remark 6.7).

<sup>1</sup> This example, and several other examples in this paper, involve graphs. Here, the word “graph” may be read as “structure over a schema with one binary relation”. Alternatively, it may be read as “structure over a schema with one binary relation that is symmetric and irreflexive”, but, in the latter case, we only require the query algorithm to behave correctly on such graphs, and we do not require the query algorithm to distinguish such graphs from structures whose relation is not symmetric and irreflexive.

#### 4 Left Query Algorithms over $\mathbb{B}$

In this section, we investigate which classes admit a left query algorithm over  $\mathbb{B}$ . It is easy to see that every class of instances that admits a left query algorithm over  $\mathbb{B}$  is closed under homomorphic equivalence. In other words, closure under homomorphic equivalence is a necessary condition for the existence of a left query algorithm over  $\mathbb{B}$ . The next result gives an exact characterization of the classes of instances that admit a left query algorithm over  $\mathbb{B}$ .

► **Theorem 4.1.** *Let  $\mathcal{C}$  be a class of instances. Then the following statements are equivalent.*

1.  $\mathcal{C}$  admits a left query algorithm over  $\mathbb{B}$ .
2.  $\mathcal{C}$  is definable by a Boolean combination of CQs.
3.  $\mathcal{C}$  is FO-definable and closed under homomorphic equivalence.

**Proof.** We will first show the equivalence between statements (1) and (2), and then the equivalence between statements (2) and (3).

(1)  $\implies$  (2): Let a left query algorithm over  $\mathbb{B}$  for  $\mathcal{C}$  consist of  $\mathcal{F} = \{F_1, \dots, F_k\}$  and  $X \subseteq \{0, 1\}^k$ , and let  $q^{F_1}, \dots, q^{F_k}$  be the canonical conjunctive queries of  $F_1, \dots, F_k$ , respectively. For every tuple  $t = (t_1, \dots, t_k) \in X$ , define

$$\varphi_t := \bigwedge_{t_i=1} q^{F_i} \wedge \bigwedge_{t_i=0} \neg q^{F_i}.$$

Take  $\varphi := \bigvee_{t \in X} \varphi_t$ . Then  $\mathcal{C} = \text{Mod}(\varphi)$ .

(2)  $\implies$  (1): Every class  $\mathcal{C}$  defined by a conjunctive query  $q$  admits a left 1-query algorithm over  $\mathbb{B}$ . Indeed, we can pick  $\mathcal{F}$  to consist of the canonical instance  $A^q$  of  $q$ , and  $X = \{1\}$ . It follows by closure under Boolean combinations that every class defined by a Boolean combination of conjunctive queries also admits a left query algorithm over  $\mathbb{B}$ .

(2)  $\implies$  (3): This implication is immediate because CQs are first-order formulas whose truth is preserved by homomorphisms.

(3)  $\implies$  (2): We will use two results from [27] about the homomorphism preservation theorem in the finite. For instances  $A$  and  $B$ , we write  $A \leftrightarrow^n B$  to mean that  $A$  and  $B$  satisfy the same existential positive FO-sentences of quantifier rank at most  $n$ , and write  $A \equiv^n B$  to mean that  $A$  and  $B$  satisfy the same FO-sentences of quantifier rank at most  $n$ . The two results from [27] in our concerns are

- (a) Theorem 1.9: For every  $n$ , there is some  $m$  that depends on  $n$  such that for every instances  $A$  and  $B$  with  $A \leftrightarrow^m B$ , there are instances  $A'$  and  $B'$  such that  $A \leftrightarrow A'$ ,  $B \leftrightarrow B'$ , and  $A' \equiv^n B'$ .
- (b) Lemma 3.9: For every  $m$ , the equivalence relation  $A \leftrightarrow^m B$  has finitely many equivalence classes over the class of all instances.

Now, let  $\mathcal{C}$  be a class definable by a FO-sentence  $\varphi$  and closed under homomorphic equivalence  $\leftrightarrow$ . Let  $n$  be the quantifier rank of  $\varphi$ , and let  $m$  be the integer in the statement of (a). Note that  $m$  depends on  $n$  only.

We claim that  $\text{Mod}(\varphi) = \mathcal{C}$  is closed under  $\leftrightarrow^m$ . Indeed, assume that  $A$  and  $B$  are two instances such that  $A \models \varphi$  and  $A \leftrightarrow^m B$ . By (a), there are instances  $A'$  and  $B'$  such that  $A \leftrightarrow A'$ ,  $B \leftrightarrow B'$ , and  $A' \equiv^n B'$ . It follows, successively, that

$$\begin{aligned} A' &\models \varphi && \text{(since } A \models \varphi, A \leftrightarrow A', \text{ and } \mathcal{C} \text{ is closed under } \leftrightarrow), \\ B' &\models \varphi && \text{(since } \varphi \text{ has quantifier rank } n \text{ and } A' \equiv^n B'), \\ B &\models \varphi && \text{(since } B \leftrightarrow B' \text{ and } \mathcal{C} \text{ is closed under } \leftrightarrow). \end{aligned}$$

By (b), the equivalence relation  $\leftrightarrow^m$  has finitely many equivalence classes. Let  $A_1, \dots, A_k$  be representatives from each of these equivalence classes (one per equivalence class). For different  $i, j$  in  $\{1, \dots, k\}$ , let  $\psi_{i,j}$  be an existential positive FO-sentence of quantifier rank at most  $m$  such that  $A_i \models \psi_{i,j}$  but not  $A_j \models \psi_{i,j}$ , and let  $\psi(A_i)$  be the conjunction of all  $\psi_{i,j}$ . Each  $\psi(A_i)$  is a Boolean combination of conjunctive queries and it holds for every instance  $B$  that  $B \leftrightarrow^m A_i$  if and only if  $B \models \psi(A_i)$ . Then  $\varphi$  is equivalent to the disjunction  $\bigvee_{A_i \models \varphi} \psi(A_i)$  since  $\text{Mod}(\varphi)$  is closed under  $\leftrightarrow^m$ . Indeed, if  $B \models \varphi$ , then  $B \leftrightarrow^m A_i$  for some  $A_i$  that satisfies  $\varphi$ , hence  $B \models \psi(A_i)$ . Conversely, if  $B \models \bigvee_{A_i \models \varphi} \psi(A_i)$ , then  $B \models \psi(A_i)$  for some  $A_i$  that satisfies  $\varphi$ ; it follows that  $B \leftrightarrow^m A_i$ , hence  $B \models \varphi$ .  $\blacktriangleleft$

► **Corollary 4.2.** *A class  $\mathcal{C}$  of instances that is closed under homomorphic equivalence admits a left query algorithm over  $\mathbb{B}$  if and only if  $\mathcal{C}$  is FO-definable.*

Corollary 4.2, in particular, applies to classes of the form  $\text{CSP}(A)$ , since such classes are closed under homomorphic equivalence. It was shown in [28] that testing, for a given instance  $A$ , whether  $\text{CSP}(A)$  is FO-definable, is NP-complete (and in fact, in polynomial time when  $A$  is a *core*, i.e. when there is no homomorphism from  $A$  to a proper subinstance of  $A$ ). It follows that testing if a given  $\text{CSP}(A)$  admits a left query algorithm over  $\mathbb{B}$  is NP-complete. This extends to finite unions of CSPs:

► **Proposition 4.3.** *The following problem is NP-complete: given instances  $A_1, \dots, A_n$ , does  $\bigcup_{1 \leq i \leq n} \text{CSP}(A_i)$  admit a left query algorithm over  $\mathbb{B}$ ?*

**Proof.** Without loss of generality, assume that the instances  $A_1, \dots, A_n$  are pairwise homomorphically incomparable (because, if  $A_j \rightarrow A_k$ , then  $A_j$  can be removed without affecting the class defined by  $\bigcup_{1 \leq i \leq n} \text{CSP}(A_i)$ ). It is known that, in this case,  $\bigcup_{1 \leq i \leq n} \text{CSP}(A_i)$  is FO-definable if and only if for each  $1 \leq i \leq n$ ,  $\text{CSP}(A_i)$  is FO-definable (this may be considered folklore, see [5, Lemma 5.13] for an explicit proof). The result follows, by Corollary 4.2.  $\blacktriangleleft$

Corollary 4.2 also applies to classes of the form  $[A]_{\leftrightarrow}$ , and we can derive a similar complexity bound. This will be obtained using the following proposition.

► **Proposition 4.4.** *Let  $A$  be an instance and  $K \in \{\mathbb{B}, \mathbb{N}\}$ . Then the following statements are equivalent.*

1.  $[A]_{\leftrightarrow}$  has a left query algorithm over  $K$ .
2.  $\text{CSP}(A)$  has a left query algorithm over  $K$ .

**Proof.** Let  $K \in \{\mathbb{B}, \mathbb{N}\}$  throughout the proof.

(2)  $\implies$  (1): Clearly,  $[A]_{\leftrightarrow} = \text{CSP}(A) \cap \{B : A \rightarrow B\}$ . Also,  $\{B : A \rightarrow B\}$  admits an obvious left query algorithm over  $K$ . The result follows by closure under Boolean combinations.

(1)  $\implies$  (2): Assume  $\text{CSP}(A)$  has no left query algorithm over  $K$ . Let  $\mathcal{F}$  be an arbitrary finite set of connected instances (think: candidate left query algorithm for  $[A]_{\leftrightarrow}$ ). Since  $\text{CSP}(A)$  has no left query algorithm over  $K$ , there are instances  $P \in \text{CSP}(A)$  and  $Q \notin \text{CSP}(A)$  such that  $\text{hom}_K(\mathcal{F}, P) = \text{hom}_K(\mathcal{F}, Q)$ . Let  $P' = P \oplus A$  and let  $Q' = Q \oplus A$ . Then, by construction,  $P' \in [A]_{\leftrightarrow}$  and  $Q' \notin [A]_{\leftrightarrow}$  but  $\text{hom}_K(\mathcal{F}, P') = \text{hom}_K(\mathcal{F}, Q')$  (cf. Proposition 2.1). Therefore, by Proposition 3.2,  $[A]_{\leftrightarrow}$  has no left query algorithm over  $K$ .  $\blacktriangleleft$

Consequently, the following problem is also NP-complete: *given an instance  $A$ , does  $[A]_{\leftrightarrow}$  admit a left query algorithm over  $\mathbb{B}$ ?*

► **Remark 4.5.** In Example 3.6, we asserted that the class  $\text{CSP}(K_3)$  of 3-colorable graphs admits no left query algorithm over  $\mathbb{B}$ . Furthermore, in Example 3.7, we asserted that  $[K_3]_{\leftrightarrow}$  (i.e., the class of graphs that are 3-colorable and also contain a triangle) admits no left query

## 8:10 When Do Homomorphism Counts Help in Query Algorithms?

algorithm over  $\mathbb{B}$ . Corollary 4.2 and Proposition 4.4, now account for the non-existence of a left query algorithm over  $\mathbb{B}$  for these classes: the reason is these two classes are not FO-definable. In the next section (see Corollary 5.6), we will see that the same explanation accounts for the fact that these classes admit no left query algorithm over  $\mathbb{N}$  either.

Proposition 4.4 extends to finite unions of homomorphic equivalence classes.

- **Theorem 4.6.** *For all instances  $A_1, \dots, A_n$ , the following statements are equivalent.*
1.  $\bigcup_{1 \leq i \leq n} [A_i]_{\leftrightarrow}$  admits a left query algorithm over  $\mathbb{B}$  (equivalently, is FO-definable).
  2. Each  $[A_i]_{\leftrightarrow}$ , for  $i = 1, \dots, n$ , admits a left query algorithm over  $\mathbb{B}$  (equivalently, is FO-definable).

**Proof.** It is clear that the second statement implies the first. We will prove by induction on  $n$  that the first statement implies the second. The base case ( $n = 1$ ) is immediate since (i) and (ii) coincide. Next, let  $n > 1$  and  $\mathcal{C} := \bigcup_{1 \leq i \leq n} [A_i]_{\leftrightarrow}$ . We proceed by contraposition: suppose that  $[A_i]_{\leftrightarrow}$  does not admit a left query algorithm over  $\mathbb{B}$ , for some  $i \leq n$ . We may assume without loss of generality that  $A_1, \dots, A_n$  are pairwise not homomorphically equivalent. Note that  $\rightarrow$  induces a preorder among  $A_1, \dots, A_n$  and, since  $n$  is finite, there is a maximal. Without loss of generality, assume that  $A_n$  is a maximal, that is,  $A_n \not\rightarrow A_i$  for all  $i < n$ . We distinguish two cases.

(1)  $[A_n]_{\leftrightarrow}$  admits a left query algorithm over  $\mathbb{B}$ . Then, for some  $i \leq n-1$ ,  $[A_i]_{\leftrightarrow}$  does not admit a left query algorithm over  $\mathbb{B}$ . By induction hypothesis, we have  $\mathcal{C}' := \bigcup_{1 \leq i \leq n-1} [A_i]_{\leftrightarrow}$  does not admit a left query algorithm over  $\mathbb{B}$ . Then it follows that  $\mathcal{C}$  does not admit a left query algorithm over  $\mathbb{B}$  either, for otherwise  $\mathcal{C}' = \mathcal{C} \setminus [A_n]_{\leftrightarrow}$  would admit a left query algorithm over  $\mathbb{B}$ .

(2)  $[A_n]_{\leftrightarrow}$  does not admit a left query algorithm over  $\mathbb{B}$ . By Proposition 4.4,  $\text{CSP}(A_n)$  does not admit a left query algorithm over  $\mathbb{B}$  either. Let  $\mathcal{F}$  be an arbitrary finite non-empty set of connected instances. Since  $\text{CSP}(A_n)$  does not admit a left query algorithm over  $\mathbb{B}$ , there are  $P \in \text{CSP}(A_n)$  and  $Q \notin \text{CSP}(A_n)$  such that  $\text{hom}_{\mathbb{B}}(\mathcal{F}, P) = \text{hom}_{\mathbb{B}}(\mathcal{F}, Q)$ . It follows that

- $(P \oplus A_n) \in [A_n]_{\leftrightarrow}$ , because both  $P, A_n \in \text{CSP}(A_n)$ ,
- $(Q \oplus A_n) \notin [A_n]_{\leftrightarrow}$ , because  $Q \notin \text{CSP}(A_n)$ ,
- for all  $i < n$ ,  $(Q \oplus A_n) \notin [A_i]_{\leftrightarrow}$ , because  $A_n \not\rightarrow A_i$ , and
- $\text{hom}_{\mathbb{B}}(\mathcal{F}, P \oplus A_n) = \text{hom}_{\mathbb{B}}(\mathcal{F}, Q \oplus A_n)$ , because the instances in  $\mathcal{F}$  are all connected.

The first three points above give  $(P \oplus A_n) \in \mathcal{C}$  and  $(Q \oplus A_n) \notin \mathcal{C}$ . Therefore, by Proposition 3.2,  $\mathcal{C}$  has no left query algorithm over  $\mathbb{B}$ . ◀

Note that Theorem 4.6 only applies to finite unions of homomorphic-equivalence classes. It may fail for infinite unions. Specifically, the class of all instances trivially admits a left query algorithm over  $\mathbb{B}$ , and it is the union of all equivalence classes  $[A]_{\leftrightarrow}$ , as  $A$  varies over all instances; however, as seen earlier,  $[K_3]_{\leftrightarrow}$  does not admit any left query algorithm over  $\mathbb{B}$ .

- **Corollary 4.7.** *The following problem is NP-complete: given instance  $A_1, \dots, A_n$ , does  $\bigcup_{1 \leq i \leq n} [A_i]_{\leftrightarrow}$  admit a left query algorithm over  $\mathbb{B}$ ?*

Each class  $\mathcal{C}$  that is closed under homomorphic equivalence can trivially be represented as a possibly-infinite union of classes of the form  $[A]_{\leftrightarrow}$ . The algorithmic problem of testing for the existence of a left query algorithm, of course, makes sense only for finitely presented inputs. This motivates the above corollary.

As a last case study, consider Boolean Datalog programs, i.e., Datalog programs with a zero-ary goal predicate. We omit a detailed definition of the syntax and semantics of Datalog, which can be found, e.g., in [1]. Each Datalog program  $P$  naturally defines a class of instances  $\mathcal{C}_P$ . It is well known that *the class  $\mathcal{C}_P$  is closed under homomorphic equivalence.*



Furthermore,  $\mathcal{C}_P$  is FO-definable if and only if  $P$  is “bounded” (meaning that there is a fixed number  $n$ , depending only on  $P$  and not on the input instance, such that  $P$  reaches its fixed point after at most  $n$  iterations), as was first shown by [2] and also follows from [27]. The boundedness problem for Boolean Datalog programs is undecidable [14]. Therefore, we have the following result.

► **Corollary 4.8.** *The following problem is undecidable: given a Boolean Datalog program  $P$ , does  $\mathcal{C}_P$  admit a left query algorithm over  $\mathbb{B}$ .*

## 5 Existence vs. Counting: When Does Counting Not Help?

Left query algorithms over  $\mathbb{N}$  are more powerful than left query algorithms over  $\mathbb{B}$ . This is trivially so because query algorithms over  $\mathbb{B}$  cannot distinguish homomorphically equivalent instances.

► **Example 5.1.** Let  $\mathcal{C}$  be the isomorphism class of the instance  $A$  consisting of the fact  $R(a, a)$  (in other words, the single-node reflexive digraph). Since  $\mathcal{C}$  is not closed under homomorphism equivalence, it does not admit a left query algorithm (nor a right query algorithm) over  $\mathbb{B}$ . On the other hand, it admits a left query algorithm over  $\mathbb{N}$ : by counting the number of homomorphisms from  $A$  to a given input instance  $B$ , we can verify that  $B$  contains a reflexive node; and by counting the number of homomorphisms from the instance  $A'$  consisting of a single edge  $R(a, b)$ , we can verify that the total number of edges in the graph is equal to 1. More precisely, let  $\mathcal{F} = \{A, A'\}$  and let  $X = \{(1, 1)\}$ . Then  $(\mathcal{F}, X)$  is a left query algorithm over  $\mathbb{N}$  for  $\mathcal{C}$ .

As it turns out, in a precise sense, this is the only reason why left query algorithms over  $\mathbb{N}$  are more powerful than left query algorithms over  $\mathbb{B}$ : the ability to count does not give more power when it comes to classes that are closed under homomorphic equivalence. This follows from the next theorem.

► **Theorem 5.2.** *Let  $\mathcal{C}$  be a class of instances closed under homomorphic equivalence. For every finite set  $\mathcal{F}$  of connected instances, the following statements are equivalent.*

1.  $\mathcal{C}$  admits a left query algorithm over  $\mathbb{N}$  of the form  $(\mathcal{F}, X)$  for some set  $X$ .
2.  $\mathcal{C}$  admits a left query algorithm over  $\mathbb{B}$  of the form  $(\mathcal{F}, X')$  for some set  $X'$ .

**Proof.** The case  $\mathcal{C} = \emptyset$  is trivial, so we will assume that  $\mathcal{C}$  is non-empty. The implication (2)  $\implies$  (1) is given by Proposition 3.3. Let us prove the implication (1)  $\implies$  (2).

Let  $(\mathcal{F}, X)$  be a left query algorithm over  $\mathbb{N}$  for  $\mathcal{C}$  where  $\mathcal{F} = \{F_1, \dots, F_k\}$  of pairwise non-isomorphic instances and  $X \subseteq \mathbb{N}^k$ . It is enough to focus on *simple* sets  $X$ , where a set  $X$  is simple if for every  $1 \leq i \leq k$  and every  $\mathbf{t}, \mathbf{t}' \in X$ ,  $\mathbf{t}(i) = 0$  iff  $\mathbf{t}'(i) = 0$ . Indeed, assume that (1)  $\implies$  (2) holds whenever  $X$  is simple. Then, if  $X$  is not simple, partition  $X$  into maximal simple subsets  $X_1, \dots, X_r$  and, for every  $1 \leq i \leq r$ , let  $\mathcal{C}_i$  be the class of instances that admits the left query algorithm  $(\mathcal{F}, X_i)$ . It is easy to verify that  $\mathcal{C}_i$  is closed under homomorphic equivalence and, hence, by assumption, admits a left query algorithm over  $\mathbb{B}$  of the form  $(\mathcal{F}, X'_i)$  for some set  $X'_i$ . Then  $(\mathcal{F}, \bigcup_{1 \leq i \leq r} X'_i)$  is a left query algorithm over  $\mathbb{B}$  for  $\mathcal{C}$ .

Let us assume that  $X$  is simple and non-empty (otherwise  $\mathcal{C} = \emptyset$ , contradicting our assumption) and let  $\mathbf{t} \in X$ . We can assume, by reordering the instances in  $\mathcal{F}$  if necessary, that there exists  $s \geq 0$ , such that  $\mathbf{t}(i) > 0$  for every  $i \leq s$  and  $\mathbf{t}(i) = 0$  for every  $i > s$ .



## 8:12 When Do Homomorphism Counts Help in Query Algorithms?

Consider the FO-sentence defined as:

$$\varphi = \bigwedge_{i \leq s} q^{F_i} \wedge \bigwedge_{i > s} \neg q^{F_i}$$

We shall prove that  $\text{Mod}(\varphi) = \mathcal{C}$ , and therefore  $\mathcal{C}$  admits a left query algorithm over  $\mathbb{B}$  (namely, the left query algorithm  $(\mathcal{F}, \{\mathbf{t}'\})$  where  $\mathbf{t}'(i) = 1$  for  $i \leq s$  and  $\mathbf{t}'(i) = 0$  for  $i > s$ ). Since  $\text{Mod}(\varphi) = \mathcal{C}$  already holds when  $s = 0$ , we assume  $s > 0$  in the sequel.

To this end we shall need the following two technical lemmas.

► **Lemma 5.3.** *Let  $\mathbf{t}_1, \dots, \mathbf{t}_r \in \mathbb{N}^s$  satisfying the following conditions:*

1. *For every  $1 \leq i \leq s$ , there exists  $1 \leq j \leq r$  such that  $\mathbf{t}_j(i) \neq 0$ .*
2. *For every different  $i, i' \in \{1, \dots, s\}$ , there exists  $1 \leq j \leq r$  such that  $\mathbf{t}_j(i) \neq \mathbf{t}_j(i')$ .*

*Then there exists some integer  $c > 0$  such that for every  $\mathbf{b} \in \mathbb{Z}^s$  whose entries are integers divisible by  $c$ , there is a multivariate polynomial with integer coefficients  $p(x_1, \dots, x_r)$  of degree  $s$  satisfying  $p(0, \dots, 0) = 0$  (that is, such that the independent term is zero) and such that, for each  $1 \leq i \leq s$ ,  $p(\mathbf{t}_1(i), \dots, \mathbf{t}_r(i)) = \mathbf{b}(i)$ .*

**Proof.** Most likely this follows from known results but, in any case, we provide a self-contained proof. Let  $\mathcal{G}$  be the set of all linear combinations  $a_1 \mathbf{t}_1 + \dots + a_r \mathbf{t}_r$  where each  $a_i$  is a non-negative integer. It follows from condition (1) that  $\mathcal{G}$  contains a tuple where all entries are positive. Moreover it follows from (2) that  $\mathcal{G}$  contains a tuple where all entries are different and positive. Indeed, if  $\mathbf{t}, \mathbf{u} \in \mathcal{G}$  and  $\mathbf{v} = d\mathbf{t} + \mathbf{u}$  for  $d \in \mathbb{N}$  large enough, then for every different  $i, i' \in \{1, \dots, s\}$ , we have

$$(\mathbf{t}(i) \neq \mathbf{t}(i') \text{ or } \mathbf{u}(i) \neq \mathbf{u}(i')) \text{ implies } \mathbf{v}(i) \neq \mathbf{v}(i').$$

Now, let  $\mathbf{u} = a_1 \mathbf{t}_1 + \dots + a_r \mathbf{t}_r$  be any tuple where all entries are different and positive. It is easy to see that the  $(s \times s)$ -matrix  $M$  whose rows are  $\mathbf{u}^1, \mathbf{u}^2, \dots, \mathbf{u}^s$  is non-singular. Indeed, if  $\mathbf{u} = (u_1, \dots, u_s)$ , then  $\det(M) = u_1 \cdots u_s \cdot \det(N)$ , where  $N$  is the  $(s \times s)$ -matrix with rows  $\mathbf{u}^0, \mathbf{u}^1, \dots, \mathbf{u}^{s-1}$  which is Vandermonde. It is well known that Vandermonde matrices are non-singular whenever  $\mathbf{u}$  has no repeated elements (see [18] for example).

To finish the proof we choose  $c$  to be  $|\det(M)|$ . By assumption all entries of  $\mathbf{b}$  are divisible by  $c$  which implies that all entries of  $M^{-1}\mathbf{b}$  are integers, that is,  $\mathbf{b}$  can be expressed as  $e_1 \mathbf{u}^1 + \dots + e_s \mathbf{u}^s$  for some  $e_1, \dots, e_s \in \mathbb{Z}$ . Hence, the polynomial  $p(x_1, \dots, x_r) = e_1 y^1 + \dots + e_s y^s$  where  $y = a_1 x_1 + \dots + a_r x_r$  satisfies the claim. ◀

► **Lemma 5.4.** *Let  $F, F'$  be instances such that there is no surjective homomorphism from  $F$  onto  $F'$ . Then there exists some subinstance  $H$  of  $F'$  such that  $\text{hom}_{\mathbb{N}}(F, H) \neq \text{hom}_{\mathbb{N}}(F', H)$ .*

**Proof.** This is a natural adaptation of the Lovász's proof that two instances are isomorphic if and only if they have the same left homomorphism-count vector.

For every instance  $G$ , we write  $\text{sur}_{\mathbb{N}}(G, F')$  for the number of surjective homomorphisms from  $G$  onto  $F'$ ; moreover, for every subset  $S \subseteq \text{adom}(F')$ , we write  $\text{hom}_{\mathbb{N}}^S(G, F')$  for the number of homomorphisms  $h : G \rightarrow F'$  whose range is contained in  $S$ . Let  $n := |\text{adom}(F')|$ . By the Inclusion-Exclusion Principle, then

$$\text{sur}_{\mathbb{N}}(G, F') = \sum_{S \subseteq \text{adom}(F')} (-1)^{n-|S|} \text{hom}_{\mathbb{N}}^S(G, F').$$

Since  $\text{sur}_{\mathbb{N}}(F, F') = 0$  and  $\text{sur}_{\mathbb{N}}(F', F') > 0$ , we have  $\text{sur}_{\mathbb{N}}(F, F') \neq \text{sur}_{\mathbb{N}}(F', F')$  and it follows by the above discussion that there is a subset  $S \subseteq \text{adom}(F')$  such that  $\text{hom}_{\mathbb{N}}^S(F, F') \neq \text{hom}_{\mathbb{N}}^S(F', F')$ . Let  $H$  be the maximum subinstance of  $F'$  with  $\text{adom}(H) \subseteq S$ , then we have  $\text{hom}_{\mathbb{N}}(F, H) = \text{hom}_{\mathbb{N}}^S(F, F') \neq \text{hom}_{\mathbb{N}}^S(F', F') = \text{hom}_{\mathbb{N}}(F', H)$ . ◀

Let us now continue with the proof that  $\text{Mod}(\varphi) = \mathcal{C}$ . The direction  $\supseteq$  is immediate. For the converse, we must prove that every  $B \in \text{Mod}(\varphi)$  belongs also to  $\mathcal{C}$ . Let  $B \in \text{Mod}(\varphi)$ , and choose an arbitrary  $A \in \mathcal{C}$  (by the assumption that  $\mathcal{C} \neq \emptyset$ ). Let  $\mathbf{t}^A = \text{hom}_{\mathbb{N}}(\mathcal{F}, A)$  and  $\mathbf{t}^B = \text{hom}_{\mathbb{N}}(\mathcal{F}, B)$ . Note that  $\mathbf{t}^A(i) = \mathbf{t}^B(i) = 0$  for every  $i > s$  and  $\mathbf{t}^A(i)$  and  $\mathbf{t}^B(i)$  are strictly positive for every  $i \leq s$ .

Let  $\mathcal{H} = \{H_1, \dots, H_r\}$  be the non-empty set of instances constructed as follows:

- (i) For every  $1 \leq i \leq s$ ,  $F_i$  is contained in  $\mathcal{H}$ .
- (ii) For every  $i, i' \in \{1, \dots, s\}$  such that there is no surjective homomorphism  $F_i \rightarrow F_{i'}$ ,  $\mathcal{H}$  contains the instance  $H$  given by Lemma 5.4.

For each  $1 \leq i \leq r$ , let  $\mathbf{t}_i = \text{hom}_{\mathbb{N}}(\{F_1, \dots, F_s\}, H_i)$ . It can be readily verified that  $\mathbf{t}_1, \dots, \mathbf{t}_r$  satisfy the conditions of Lemma 5.3. Condition (1) is guaranteed due to step (i) in the construction of  $\mathcal{H}$ . For condition (2), let  $i, i'$  be any pair of different integers in  $\{1, \dots, s\}$ . Since  $F_i$  and  $F_{i'}$  are not isomorphic, it must be the case that there is no surjective homomorphism  $F_i \rightarrow F_{i'}$  or there is no surjective homomorphism  $F_{i'} \rightarrow F_i$ . Hence, due to step (ii),  $\mathcal{H}$  contains some instance  $H_j$  witnessing  $\mathbf{t}_j(i) \neq \mathbf{t}_j(i')$ .

Let  $c > 0$  be given by Lemma 5.3, let  $\mathbf{b} \in \mathbb{Z}^s$  where  $\mathbf{b}(i) = c(\mathbf{t}^B(i) - \mathbf{t}^A(i))$  for every  $1 \leq i \leq s$ , and let  $p(x_1, \dots, x_r)$  be the polynomial given by Lemma 5.3 for  $\mathbf{b}$ . We can express  $p(x_1, \dots, x_r)$  as  $p_A(x_1, \dots, x_r) - p_B(x_1, \dots, x_r)$  where all the coefficients in  $p_A$  and  $p_B$  are positive.

For every polynomial  $q(x_1, \dots, x_r)$  where the independent term is zero, consider the instance  $H_q$  defined inductively as follows:

- If  $q = x_j$ , then  $H_q$  is  $H_j$ .
- If  $q = u + v$ , then  $H_q$  is  $H_u \oplus H_v$ .
- If  $q = u \cdot v$ , then  $H_q = H_u \otimes H_v$ .

► **Lemma 5.5.** *Let  $H_q$  be constructed as above. Then:*

1.  $H_q \rightarrow F_1 \oplus \dots \oplus F_s$ .
2. Let  $F$  be a connected instance and let  $\mathbf{t} = \text{hom}_{\mathbb{N}}(F, \mathcal{H})$ . Then  $\text{hom}_{\mathbb{N}}(F, H_q) = q(\mathbf{t})$ .

This lemma directly follows from the definition of  $H_q$ . More precisely, the first item follows from the fact that  $H_j \rightarrow F_1 \oplus \dots \oplus F_s$  for every  $H_j \in \mathcal{H}$  (as can be shown by a straightforward induction argument). The second item follows from the definition of  $H_q$  and Proposition 2.1.

Let  $A' = A_1 \oplus \dots \oplus A_c \oplus H_{p_A}$  where  $A_1, \dots, A_c$  are disjoint copies of  $A$ . Similarly, define  $B' = B_1 \oplus \dots \oplus B_c \oplus H_{p_B}$  where  $B_1, \dots, B_c$  are disjoint copies of  $B$ . We claim that  $\mathbf{t}^{A'} = \text{hom}_{\mathbb{N}}(\mathcal{F}, A')$  and  $\mathbf{t}^{B'} = \text{hom}_{\mathbb{N}}(\mathcal{F}, B')$  coincide.

Let  $1 \leq i \leq k$  and consider first the case  $i > s$ . It follows from Lemma 5.5(1) that  $F_i \not\rightarrow H_{p_A}$ . Indeed, if  $F_i \rightarrow H_{p_A}$ , then  $F_i \rightarrow F_{i'}$  for some  $i' \leq s$  implying that  $\mathcal{C} = \emptyset$ , contradicting our assumption. Similarly  $F_i \not\rightarrow H_{p_B}$ . Consequently,  $\mathbf{t}^{A'}(i) = c \cdot \mathbf{t}^A(i) = 0 = c \cdot \mathbf{t}^B(i) = \mathbf{t}^{B'}(i)$ . For the other case, namely  $i \leq s$ , we have

$$\begin{aligned}
\mathbf{t}^{B'}(i) - \mathbf{t}^{A'}(i) &= \text{hom}_{\mathbb{N}}(F_i, B') - \text{hom}_{\mathbb{N}}(F_i, A') \\
&= c \cdot \text{hom}_{\mathbb{N}}(F_i, B) - c \cdot \text{hom}_{\mathbb{N}}(F_i, A) + \text{hom}_{\mathbb{N}}(F_i, H_{p_B}) - \text{hom}_{\mathbb{N}}(F_i, H_{p_A}) \\
&= c \cdot (\mathbf{t}^B(i) - \mathbf{t}^A(i)) + \text{hom}_{\mathbb{N}}(F_i, H_{p_B}) - \text{hom}_{\mathbb{N}}(F_i, H_{p_A}) \\
&= c \cdot (\mathbf{t}^B(i) - \mathbf{t}^A(i)) + p_B(\mathbf{t}_1(i), \dots, \mathbf{t}_r(i)) - p_A(\mathbf{t}_1(i), \dots, \mathbf{t}_r(i)) \\
&= c \cdot (\mathbf{t}^B(i) - \mathbf{t}^A(i)) - p(\mathbf{t}_1(i), \dots, \mathbf{t}_r(i)) \\
&= c \cdot (\mathbf{t}^B(i) - \mathbf{t}^A(i)) - \mathbf{b}(i) = 0,
\end{aligned}$$

where  $\text{hom}_{\mathbb{N}}(F_i, H_{p_A}) = p_A(\mathbf{t}_1(i), \dots, \mathbf{t}_r(i))$  and  $\text{hom}_{\mathbb{N}}(F_i, H_{p_B}) = p_B(\mathbf{t}_1(i), \dots, \mathbf{t}_r(i))$  hold by Lemma 5.5(2).

## 8:14 When Do Homomorphism Counts Help in Query Algorithms?

To finish the proof, note that since  $\mathbf{t}^A(i) > 0$  for every  $1 \leq i \leq s$  it follows by Lemma 5.5(1) that  $A' \rightarrow A$ , and, hence  $A \leftrightarrow A'$ . Similarly we have  $B' \leftrightarrow B$ . Since  $\mathcal{C}$  is closed under homomorphic equivalence we have that  $A' \in \mathcal{C}$ . Since  $\mathbf{t}^{A'} = \mathbf{t}^{B'}$  it follows that  $B'$  and, hence,  $B$  belong to  $\mathcal{C}$  as well.  $\blacktriangleleft$

► **Corollary 5.6.** *Let  $\mathcal{C}$  be a class of instances closed under homomorphic equivalence. Then the following statements are equivalent.*

1.  $\mathcal{C}$  admits a left query algorithm over  $\mathbb{N}$ .
2.  $\mathcal{C}$  admits a left query algorithm over  $\mathbb{B}$ .
3.  $\mathcal{C}$  is FO-definable.

The equivalence of statements (1) and (2) follows from Theorem 5.2 together with Proposition 3.2. The equivalence of statements (2) and (3) was already established in Corollary 4.2.

We would like to point out some interesting special cases of Theorem 5.2. The first pertains to CSPs. Let us say that a constraint satisfaction problem  $\text{CSP}(B)$  is “*determined by*  $\text{hom}_K(\mathcal{F}, \cdot)$ ” for some  $K \in \{\mathbb{B}, \mathbb{N}\}$  and some class  $\mathcal{F}$  of instances, if the following holds for all instances  $A$  and  $A'$ : if  $\text{hom}_K(\mathcal{F}, A) = \text{hom}_K(\mathcal{F}, A')$  then  $A \in \text{CSP}(B)$  iff  $A' \in \text{CSP}(B)$ . Then Theorem 5.2 can be rephrased as follows: for every finite set of connected instances  $\mathcal{F}$ , every CSP determined by  $\text{hom}_{\mathbb{N}}(\mathcal{F}, \cdot)$  is determined by  $\text{hom}_{\mathbb{B}}(\mathcal{F}, \cdot)$ . In contrast, for  $\mathcal{T}$  the infinite class of all trees, the CSPs determined by  $\text{hom}_{\mathbb{B}}(\mathcal{T}, \cdot)$  form a proper subclass of those determined by  $\text{hom}_{\mathbb{N}}(\mathcal{T}, \cdot)$ .<sup>2</sup> This follows from results in [8], because the former are precisely the CSPs that can be solved using arc-consistency, while the latter are precisely the CSPs that can be solved using basic linear programming relaxation (BLP). See [23, Example 99] for an example of a CSP that can be solved using BLP but not using arc-consistency.

The second special case pertains to homomorphic-equivalence classes. Given the importance of homomorphic equivalence as a notion of equivalence in database theory, it is natural to ask when a database instance  $A$  can be uniquely identified up to homomorphic equivalence by means of a left query algorithm. As we saw earlier, in Section 4, for left query algorithm over  $\mathbb{B}$ , this is the case if and only if  $\text{CSP}(A)$  is FO-definable (a condition that can be tested effectively, and, in fact, is NP-complete to test). It follows from Corollary 5.6 that the same criterion determines whether  $A$  can be uniquely identified up to homomorphic equivalence by means of a left query algorithm over  $\mathbb{N}$ . This extends naturally to finite unions of homomorphic equivalence classes.

Finally, let us consider again classes  $\mathcal{C}$  defined by a Boolean Datalog program  $P$ . It follows from the results we mentioned in Section 4 that such a class of instances  $\mathcal{C}$  admits a left query algorithm over  $\mathbb{N}$  if and only if  $P$  is bounded, and that testing for the existence of a left query algorithm over  $\mathbb{N}$  is undecidable.

## 6 Right Query Algorithms

Just as for left query algorithms, we have that every class of instances that admits a right query algorithm over  $\mathbb{B}$  is closed under homomorphic equivalence. However, *unlike* left query algorithms, a class of instances that admits a right query algorithm over  $\mathbb{B}$  is not necessarily FO-definable. Concretely, as we saw in Example 3.6, the class of 3-colorable graphs admits a right query algorithm over  $\mathbb{B}$ , but is not FO-definable. In fact, *every* constraint satisfaction problem  $\text{CSP}(A)$  admits a right query algorithm over  $\mathbb{B}$ , and the FO-definable ones are precisely those that admit a left query algorithm over  $\mathbb{B}$ . The next result is straightforward.

<sup>2</sup> Note that the definitions of  $\text{hom}_{\mathbb{N}}(\mathcal{F}, \cdot)$  and  $\text{hom}_{\mathbb{B}}(\mathcal{F}, \cdot)$  extend naturally to infinite classes  $\mathcal{F}$ .

► **Proposition 6.1.** *Let  $\mathcal{C}$  be a class of instances. Then  $\mathcal{C}$  admits a right query algorithm over  $\mathbb{B}$  if and only if  $\mathcal{C}$  is definable by a Boolean combination of CSPs.*

We also saw in Section 3 that not every homomorphic-equivalence closed FO-definable class admits a right query algorithm over  $\mathbb{B}$ . For example, the class of triangle-free graphs does not admit a right query algorithm over  $\mathbb{B}$  (or even over  $\mathbb{N}$ ). This raises the question which homomorphic-equivalence closed FO-definable classes admit a right query algorithm over  $\mathbb{B}$ . Equivalently, *which classes  $\mathcal{C}$  that admit a left query algorithm over  $\mathbb{B}$ , admit a right query algorithm over  $\mathbb{B}$ ?* We will address this question next by making use of two known results.

► **Theorem 6.2** (Sparse Incomparability Lemma, [22]). *Let  $m, n \geq 0$ . For every instance  $B$  there is an instance  $B^*$  of girth at least  $m$  such that, for all instances  $D$  with  $|\text{adom}(D)| \leq n$ , we have  $B \in \text{CSP}(D)$  if and only if  $B^* \in \text{CSP}(D)$ .*

A *finite homomorphism duality* is a pair of finite sets  $(\mathcal{F}, \mathcal{D})$  such that, for every instance  $A$ , the following are equivalent: (i)  $F \rightarrow A$  for some  $F \in \mathcal{F}$ ; (ii)  $A \not\rightarrow D$  for all  $D \in \mathcal{D}$ . We make use of the following known characterization of this notion.

► **Theorem 6.3** ([13]). *Let  $A$  be an instance. Then the following statements are equivalent.*

1.  *$A$  is homomorphically equivalent to an acyclic instance.*
2. *There is a finite homomorphism duality  $(\mathcal{F}, \mathcal{D})$  with  $\mathcal{F} = \{A\}$ .*
3. *There is a finite homomorphism duality  $(\mathcal{F}, \mathcal{D})$  with  $\mathcal{F} = \{A\}$  and where  $\mathcal{D}$  consists of instances  $D$  for which  $\text{CSP}(D)$  is FO-definable.*

To see the implication (2)  $\implies$  (3), note that, if  $(\mathcal{F}, \mathcal{D})$  is a finite homomorphism duality, then  $\bigcup_{D \in \mathcal{D}} \text{CSP}(D)$  is a FO-definable class (because it is defined by the negation of the UCQ  $\bigvee_{F \in \mathcal{F}} q^F$ ). It follows, by the same reasoning as in the proof of Proposition 4.3, that exists  $\mathcal{D}' \subseteq \mathcal{D}$  such that  $(\mathcal{F}, \mathcal{D}')$  is a finite homomorphism duality and such that, for each  $D \in \mathcal{D}'$ , we have that  $\text{CSP}(D)$  is FO-definable.

Our next result characterizes the classes that admit both a left query algorithm and a right query algorithm over  $\mathbb{B}$ .

► **Theorem 6.4.** *Let  $\mathcal{C}$  be a class of instances. Then the following statements are equivalent.*

1.  *$\mathcal{C}$  admits both a left query algorithm and a right query algorithm over  $\mathbb{B}$ .*
2.  *$\mathcal{C}$  admits a left query algorithm over  $\mathbb{N}$  and a right query algorithm over  $\mathbb{B}$ .*
3.  *$\mathcal{C}$  is definable by a Boolean combination of Berge-acyclic CQs.*
4.  *$\mathcal{C}$  is definable by a Boolean combination of FO-definable CSPs.*

**Proof.** The equivalence of statements (1) and (2) follows from Corollary 5.6. The proof of the remaining equivalences is as follows.

(1)  $\implies$  (3) By Theorem 4.1 and Proposition 6.1,  $\mathcal{C}$  is definable by a Boolean combination  $\varphi$  of CQs, as well as by a Boolean combination  $\psi$  of CSPs. Let  $\varphi'$  be obtained from  $\varphi$  by replacing each conjunctive query  $q$  by the disjunction of all homomorphic images of  $q$  that are Berge-acyclic. We claim that  $\varphi'$  defines  $\mathcal{C}$ .

By Theorem 6.3, we know that  $\varphi'$  is also equivalent to a Boolean combination of CSPs, which we may call  $\psi'$ . Let  $n$  be the maximum size of a CSP occurring in  $\psi$  and  $\psi'$ . Also, let  $m$  be greater than the maximum size of the CQs in  $\varphi$ . Let  $B$  be any instance, and let  $B^*$  now be the instance given by Theorem 6.2 (for  $m, n$  as chosen above). By construction,  $B$  and  $B^*$  agree with each other on their membership in CSPs of size at most  $n$ , which include all CSPs occurring in  $\psi$  as well as  $\psi'$ , and therefore,  $B$  and  $B^*$  agree on  $\psi$  and  $\psi'$ . Since  $\psi$  is equivalent to  $\varphi$  and  $\psi'$  is equivalent to  $\varphi'$ , this means that  $B$  and  $B^*$  agree on  $\varphi$  and  $\varphi'$ . Furthermore, by construction,  $\varphi'$  is equivalent to  $\varphi$  on instances of girth at least  $m$  (because

## 8:16 When Do Homomorphism Counts Help in Query Algorithms?

every homomorphic image of a CQ of size less than  $m$  in such an instance must be acyclic). In particular, it follows that  $B^* \models \varphi$  iff  $B^* \models \varphi'$ . Putting everything together, we have that  $B \models \varphi$  iff  $B^* \models \varphi$  iff  $B^* \models \varphi'$  iff  $B \models \varphi'$ .

(3)  $\implies$  (4) This follows from Theorem 6.3 (for  $A$  the canonical instance of  $q$ ): we simply replace each Berge-acyclic conjunctive query  $q$  by the conjunction  $\bigwedge_{D \in \mathcal{D}} \neg \text{CSP}(D)$ .

(4)  $\implies$  (1) This follows from Theorem 4.1 and Proposition 6.1.  $\blacktriangleleft$

Let  $\mathcal{C}$  be a class that admits a left query algorithm over  $\mathbb{B}$  or, equivalently, let  $\mathcal{C}$  be definable by a Boolean combination of CQs. It follows that  $\mathcal{C}$  admits a right query algorithm over  $\mathbb{B}$  if and only if  $\mathcal{C}$  is definable by a Boolean combination of *Berge-acyclic* CQs. Similarly, let  $\mathcal{C}$  be a class that admits a right query algorithm over  $\mathbb{B}$  or, equivalently, let  $\mathcal{C}$  be definable by a Boolean combination of CSPs. It follows that  $\mathcal{C}$  admits a left query algorithm over  $\mathbb{B}$  if and only if  $\mathcal{C}$  is definable by a Boolean combination of *FO-definable* CSPs.

Finally, we will consider the question when a class of the form  $[A]_{\leftrightarrow}$  admits a right query algorithm over  $\mathbb{B}$ .

**► Theorem 6.5.** *Let  $A$  be an instance. Then the following statements are equivalent.*

1.  $[A]_{\leftrightarrow}$  has a right query algorithm over  $\mathbb{B}$ .
2.  $\{B : A \rightarrow B\}$  has a right query algorithm over  $\mathbb{B}$ .
3.  $A$  is homomorphically equivalent to an acyclic instance.

*Moreover, testing whether this holds (for a given instance  $A$ ) is NP-complete.*

**Proof.** We will close a cycle of implications.

(1)  $\implies$  (2): For this, we use the exponentiation operation  $X^Y$  [17]. This operation has the property that  $X \rightarrow Y^Z$  if and only if  $X \otimes Z \rightarrow Y$ . Assume  $\{B : A \rightarrow B\}$  does not admit a right query algorithm over  $\mathbb{B}$ . Consider an arbitrary finite set of instances  $\mathcal{F} = \{F_1, \dots, F_k\}$  (think: candidate right query algorithm for  $[A]_{\leftrightarrow}$ ). Since  $\{B : A \rightarrow B\}$  does not admit a right query algorithm over  $\mathbb{B}$ , for the set  $\mathcal{F}^A := \{F_1^A, \dots, F_k^A\}$  there are instances  $P$  and  $Q$  with  $A \rightarrow P$  and  $A \not\rightarrow Q$  such that  $\text{hom}_{\mathbb{B}}(P, \mathcal{F}^A) = \text{hom}_{\mathbb{B}}(Q, \mathcal{F}^A)$  or, equivalently,  $\text{hom}_{\mathbb{B}}(P \otimes A, \mathcal{F}) = \text{hom}_{\mathbb{B}}(Q \otimes A, \mathcal{F})$ . Let  $P' := P \otimes A$  and let  $Q' := Q \otimes A$ . Then, by Proposition 2.1,  $P' \in [A]_{\leftrightarrow}$  and  $Q' \notin [A]_{\leftrightarrow}$  but  $\text{hom}_{\mathbb{B}}(P', \mathcal{F}) = \text{hom}_{\mathbb{B}}(Q', \mathcal{F})$ . Therefore  $[A]_{\leftrightarrow}$  has no right query algorithm over  $\mathbb{B}$ .

(2)  $\implies$  (3): Assume that  $\{B \mid A \rightarrow B\}$  admits a right query algorithm for  $\mathbb{B}$ . We will construct a finite homomorphism duality  $(\mathcal{F}, \mathcal{D})$  with  $\mathcal{F} = \{A\}$ . It then follows by Theorem 6.3 that  $A$  is homomorphically equivalent to an acyclic instance. Let  $B_1, \dots, B_n$  be all those instances  $B_i$  used by the right query algorithm for which it holds that  $A \not\rightarrow B_i$ . We claim that  $(\{A\}, \{B_1, \dots, B_n\})$  is a finite homomorphism duality. Let  $C$  be any instance. If  $C \rightarrow B_i$  for some  $i \leq n$ , then  $A \not\rightarrow C$  (otherwise, by transitivity, we would have that  $A \rightarrow B_i$ ). Conversely, if  $A \not\rightarrow C$ , then the algorithm must answer “no” on input  $C$  while it answers “yes” on input  $C \oplus A$ . Therefore, one of the right-queries made by the algorithm must differentiate  $C$  from  $C \oplus A$ . It is easy to see that the right-query in question must consist of an instance into which  $C$  maps but  $A$  does not. This instance must then be among the  $B_i$ , and  $C \rightarrow B_i$ .

(3)  $\implies$  (1): By Theorem 6.3, there is a finite homomorphism duality  $(\{A\}, \mathcal{D})$ . In particular, for all instances  $C$ , we have that  $C \in [A]_{\leftrightarrow}$  if and only if  $C \rightarrow A$  and  $C \not\rightarrow D$  for all  $D \in \mathcal{D}$ .

To test if a given instance is homomorphically equivalent to an acyclic instance, it suffices to test that its core is acyclic (equivalently, that it has an acyclic retract). This can clearly be done in NP. The NP-hardness follows directly from Theorem 6 in [10].  $\blacktriangleleft$

The preceding Theorem 6.5 can be thought of as an analogue of Proposition 4.4 for right query algorithms. Again, this result extends to finite unions of homomorphism-equivalence classes.

► **Theorem 6.6.** *For all instances  $A_1, \dots, A_n$ , the following statements are equivalent.*

1.  $\bigcup_{1 \leq i \leq n} [A_i]_{\leftrightarrow}$  admits a right query algorithm over  $\mathbb{B}$ .
  2. Each  $[A_i]_{\leftrightarrow}$ , for  $i = 1, \dots, n$ , admits a right query algorithm over  $\mathbb{B}$ .
- In particular, testing whether this holds (for given instances  $A_1, \dots, A_n$ ) is NP-complete.*

**Proof.** It is clear that the second statement implies the first. We will prove by induction on  $n$  that the first statement implies the second. The base case ( $n = 1$ ) is immediate since then statements (1) and (2) coincide. Next, let  $n > 1$  and  $\mathcal{C} := \bigcup_{1 \leq i \leq n} [A_i]_{\leftrightarrow}$ . We proceed by contraposition, assuming that  $[A_i]_{\leftrightarrow}$  does not admit a right query algorithm over  $\mathbb{B}$  for some  $i \leq n$ . We may assume without loss of generality that  $A_1, \dots, A_n$  are pairwise not homomorphically equivalent. Note that  $\rightarrow$  induces a preorder among  $A_1, \dots, A_n$  and, since  $n$  is finite, there is a minimal. Without loss of generality, assume that  $A_n$  is a minimal, that is,  $A_i \not\rightarrow A_n$  for all  $i < n$ . We distinguish two cases.

(1)  $[A_n]_{\leftrightarrow}$  admits a right query algorithm over  $\mathbb{B}$ . Then, for some  $i \leq n - 1$ ,  $[A_i]_{\leftrightarrow}$  does not admit any right query algorithm over  $\mathbb{B}$ . By induction hypothesis, we have  $\mathcal{C}' := \bigcup_{1 \leq i \leq n-1} [A_i]_{\leftrightarrow}$  does not admit any right query algorithm over  $\mathbb{B}$ . Then  $\mathcal{C}$  does not admit a right query algorithm over  $\mathbb{B}$  either, since  $\mathcal{C}' = \mathcal{C} \setminus [A_n]_{\leftrightarrow}$ .

(2)  $[A_n]_{\leftrightarrow}$  does not admit any right query algorithm over  $\mathbb{B}$ . By Theorem 6.5, the class  $\{B : A \rightarrow B\}$  does not admit any right query algorithm over  $\mathbb{B}$ , either. Consider an arbitrary finite non-empty set of instances  $\mathcal{F} = \{F_1, \dots, F_k\}$ . Since  $\{B : A \rightarrow B\}$  does not admit any right query algorithm over  $\mathbb{B}$ , for the set  $\mathcal{F}^{A_n} = \{F_1^{A_n}, \dots, F_k^{A_n}\}$  there are instances  $P$  and  $Q$  with  $A_n \rightarrow P$  and  $A_n \not\rightarrow Q$  such that  $\text{hom}_{\mathbb{B}}(P, \mathcal{F}^{A_n}) = \text{hom}_{\mathbb{B}}(Q, \mathcal{F}^{A_n})$ , which implies that  $\text{hom}_{\mathbb{B}}(P \otimes A_n, \mathcal{F}) = \text{hom}_{\mathbb{B}}(Q \otimes A_n, \mathcal{F})$ . It follows by Proposition 2.1 that

- $(P \otimes A_n) \in [A_n]_{\leftrightarrow}$  because  $A_n \rightarrow P$ ,
- $(Q \otimes A_n) \notin [A_n]_{\leftrightarrow}$  because  $A_n \not\rightarrow Q$ ,
- for all  $i < n$ ,  $(Q \otimes A_n) \notin [A_i]_{\leftrightarrow}$  because  $A_i \not\rightarrow A_n$ .

Let  $P' := P \otimes A_n$  and let  $Q' := Q \otimes A_n$ . Then the above discussion yields that  $P' \in \mathcal{C}$  and  $Q' \notin \mathcal{C}$  while  $\text{hom}_{\mathbb{B}}(P', \mathcal{F}) = \text{hom}_{\mathbb{B}}(Q', \mathcal{F})$ . Therefore,  $\mathcal{C}$  does not admit any right query algorithm over  $\mathbb{B}$ . ◀

► **Remark 6.7.** We saw in Example 3.4 that the class of triangle-free graphs, which clearly has a left query algorithm over  $\mathbb{B}$ , does not admit a right query algorithm over  $\mathbb{B}$  or over  $\mathbb{N}$ . Observe that this class is defined by the negation of the “triangle” conjunctive query  $\exists xyz(R(x, y) \wedge R(y, z) \wedge R(z, x))$ . In light of Theorem 6.4, the lack of a right query algorithm over  $\mathbb{B}$  for this class can be “explained” by the fact that this conjunctive query is not Berge-acyclic. Furthermore, in Example 3.7 we mentioned that the class  $[K_3]_{\leftrightarrow}$ , that is, the class of graphs that are 3-colorable and also contain a triangle, does not admit a right query algorithm over  $\mathbb{B}$ . This follows from Theorem 6.5.

We conclude this section with an open problem.

► **Question 6.8.** Does a suitable analogue of Theorem 5.2 hold for right query algorithms?

## 7 Summary and Discussion of Related Topics

Inspired by the work of Chen et al. [9], we extended their framework and studied various types of query algorithms, where a query algorithm for a class  $\mathcal{C}$  of instances determines whether a given input instance belongs to  $\mathcal{C}$  by making a finite number of (predetermined) queries that ask for the existence of certain homomorphisms or for the number of certain homomorphisms. Specifically, we introduced and studied *left query algorithms* and *right*



query algorithms over  $\mathbb{B}$ , as well as over  $\mathbb{N}$ . Our results delineate the differences in expressive power between these four types of query algorithms. In particular, they pinpoint when the ability to count homomorphisms is essential for the existence of left query algorithms.

**Relationship to view determinacy.** Recently, Kwiecien et al. [24] studied view determinacy under bag semantics. In particular, they obtained a decidability result for determinacy with respect to Boolean views under bag-set semantics. We will briefly describe their framework and relate it to ours. At the most abstract level, a *view* is simply a function  $f$  that takes as input a database instance. Specifically, under set semantics, every Boolean CQ specifies a view that is a function from database instances to  $\{0, 1\}$ , while, under bag-set semantics, every Boolean CQ specifies a view that is a function from database instances to  $\mathbb{N}$ . We say that a finite collection of views  $f_1, \dots, f_k$  *determines* a view  $g$ , if for all database instances  $A$  and  $B$ , if  $f_i(A) = f_i(B)$  for all  $i \leq k$ , then  $g(A) = g(B)$ . The aforementioned result from [24] asserts that the following problem is decidable: given views  $f_1, \dots, f_k$  and  $g$  specified by Boolean CQs under bag-set semantics, is  $g$  determined by  $f_1, \dots, f_k$ ?

We now describe the relationship between the above notion of view determinacy and our framework. Let  $\mathcal{C}$  be a class of instances and let  $\mathcal{F} = \{F_1, \dots, F_k\}$  be a finite set of instances. Then the following are equivalent:

1. There exists a set  $X$  such that  $(\mathcal{F}, X)$  is a left query algorithm over  $\mathbb{N}$  for  $\mathcal{C}$ ,
2.  $f_1, \dots, f_k$  determine  $g_{\mathcal{C}}$  where  $f_i(A) = \text{hom}_{\mathbb{N}}(F_i, A)$  and  $g_{\mathcal{C}}$  is the indicator function of  $\mathcal{C}$  (i.e.,  $g_{\mathcal{C}}(A) = 1$  if  $A \in \mathcal{C}$  and  $g_{\mathcal{C}}(A) = 0$  otherwise).

This tells that there are some important differences between our framework and the one in [24]: (i) when we study the existence of left query algorithms, the set  $\mathcal{F}$  is not fixed, whereas, in the view determinacy problem, the views are given as part of the input; (ii) in the view determinacy problem studied in [24], the view  $g$  is specified by a CQ with bag-set semantics, whereas in our case  $g$  is a Boolean-valued function since it is the indicator function of a class of instances, (iii) we do not assume that the class  $\mathcal{C}$  is specified by a Boolean CQ. Indeed, if  $\mathcal{C}$  were specified by a Boolean CQ  $q$ , then a left query algorithm would trivially exist, where  $\mathcal{F}$  simply consists of (the canonical instance of)  $q$  itself.

**Other semirings.** Query algorithm over  $\mathbb{B}$  and query algorithm over  $\mathbb{N}$  can be viewed as special cases of a more general setting, namely that of a query algorithm over a semiring. There is a body of research in the interface of databases and semirings, including the study of provenance of database queries using semirings [16, 19], the study of the query containment problem under semiring semantics [15, 21], and, more recently, the study of Datalog under semiring semantics [20]. In these studies, the semirings considered are *positive*, which means that they are commutative semirings with no zero divisors and with the property that the sum of any two non-zero elements is non-zero. It is perfectly meaningful to define homomorphism counts over positive semirings and then investigate query algorithms over such semirings. In particular, it may be interesting to investigate query algorithms over the *tropical semiring*  $\mathbb{R} = (R \cup \{\infty\}, \min, +)$ , where  $R$  is the set of real numbers, since it is well known that various *shortest-distance* problems can be naturally captured using this semiring.

**Adaptive query algorithms.** The query algorithms  $(\mathcal{F}, X)$  studied in this paper are *non-adaptive*, in the sense that the set  $\mathcal{F} = \{F_1, \dots, F_k\}$  is fixed up-front. In contrast, an *adaptive* query algorithm may decide the set of instances  $\mathcal{F}$  at run-time, that is to say, the choice of  $F_i$  may depend on the homomorphism-count vector for  $F_1, \dots, F_{i-1}$ . As pointed out in the Introduction, whenever a class  $\mathcal{C}$  admits an adaptive left (right) query



algorithm over  $\mathbb{B}$ , then it also admits a non-adaptive left (right) query algorithm over  $\mathbb{B}$ . Note that the most “economical” (as regards the number of instances used) non-adaptive algorithm for a class  $\mathcal{C}$  may use a larger set  $\mathcal{F}$  of instances than the adaptive one, but the number of instances used by the non-adaptive algorithm is still finite. Thus, adaptive query algorithms over  $\mathbb{B}$  do not offer higher expressive power than adaptive ones. The situation for  $\mathbb{N}$  is quite different: it was shown in [9] that *every* isomorphism-closed class of instances (in particular, every homomorphic-equivalence closed class) admits an adaptive left  $k$ -query algorithm over  $\mathbb{N}$  already for  $k = 3$ ; therefore, adaptive left query algorithms over  $\mathbb{N}$  have higher expressive power than adaptive left query algorithms over  $\mathbb{B}$ , even when it comes to homomorphic-equivalence closed classes. Switching sides, note that the class of triangle-free graphs (Example 3.4) does not admit an adaptive right query algorithm over  $\mathbb{N}$ , as was shown in [9, Proposition 8.2]; hence it is a meaningful question to ask: which homomorphic-equivalence closed classes admit an adaptive right query algorithm over  $\mathbb{N}$ ?

---

## References

- 1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*, volume 8. Addison-Wesley Reading, 1995.
- 2 Miklos Ajtai and Yuri Gurevich. Datalog vs first-order logic. *Journal of Computer and System Sciences*, 49(3):562–588, 1994. 30th IEEE Conference on Foundations of Computer Science. doi:10.1016/S0022-0000(05)80071-6.
- 3 Albert Atserias, Phokion G Kolaitis, and Wei-Lin Wu. On the expressive power of homomorphism counts. In *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–13. IEEE, 2021. doi:10.1109/LICS52264.2021.9470543.
- 4 Michal Bielecki and Jan Van den Bussche. Database interrogation using conjunctive queries. In *Database Theory - ICDT 2003, 9th Int. Conf., 2003, Proceedings*, volume 2572 of *Lecture Notes in Computer Science*, pages 259–269. Springer, 2003. doi:10.1007/3-540-36285-1\_17.
- 5 Meghyn Bienvenu, Balder Ten Cate, Carsten Lutz, and Frank Wolter. Ontology-based data access: A study through disjunctive Datalog, CSP, and MMSNP. *ACM Trans. Database Syst.*, 39(4), dec 2015. doi:10.1145/2661643.
- 6 Jan Böker, Yijia Chen, Martin Grohe, and Gaurav Rattan. The complexity of homomorphism indistinguishability. In *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019*, volume 138 of *LIPIcs*, pages 54:1–54:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.MFCS.2019.54.
- 7 Andrei A. Bulatov. A dichotomy theorem for nonuniform CSPs. In Chris Umans, editor, *58th IEEE Annual Symp. on Found. of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 319–330. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.37.
- 8 Silvia Butti and Víctor Dalmau. Fractional homomorphism, Weisfeiler-Leman invariance, and the Sherali-Adams hierarchy for the constraint satisfaction problem. In *International Symposium on Mathematical Foundations of Computer Science*, 2021. doi:10.4230/LIPIcs.MFCS.2021.27.
- 9 Yijia Chen, Jörg Flum, Mingjun Liu, and Zhiyang Xun. On algorithms based on finitely many homomorphism counts. In *47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022)*, 2022. doi:10.4230/LIPIcs.MFCS.2022.32.
- 10 Víctor Dalmau, Phokion G. Kolaitis, and Moshe Y. Vardi. Constraint satisfaction, bounded treewidth, and finite-variable logics. In *Principles and Practice of Constraint Programming - CP 2002, 8th International Conference, CP 2002, Proceedings*, volume 2470 of *Lecture Notes in Computer Science*, pages 310–326. Springer, 2002. doi:10.1007/3-540-46135-3\_21.
- 11 Holger Dell, Martin Grohe, and Gaurav Rattan. Lovász meets Weisfeiler and Leman. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018*, volume 107 of *LIPIcs*, pages 40:1–40:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.40.

- 12 Zdenek Dvorák. On recognizing graphs by numbers of homomorphisms. *J. Graph Theory*, 64(4):330–342, 2010. doi:10.1002/jgt.20461.
- 13 Jan Foniok, Jaroslav Nešetřil, and Claude Tardif. Generalised dualities and maximal finite antichains in the homomorphism order of relational structures. *European Journal of Combinatorics*, 29(4):881–899, 2008. doi:10.1016/j.ejc.2007.11.017.
- 14 Haim Gaifman, Harry Mairson, Yehoshua Sagiv, and Moshe Y. Vardi. Undecidable optimization problems for database logic programs. *J. ACM*, 40(3):683–713, jul 1993. doi:10.1145/174130.174142.
- 15 Todd J. Green. Containment of conjunctive queries on annotated relations. *Theory Comput. Syst.*, 49(2):429–459, 2011. doi:10.1007/s00224-011-9327-6.
- 16 Todd J. Green, Gregory Karvounarakis, and Val Tannen. Provenance semirings. In Leonid Libkin, editor, *Proceedings of the Twenty-Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, 2007*, pages 31–40. ACM, 2007. doi:10.1145/1265530.1265535.
- 17 Pavol Hell and Jaroslav Nešetřil. *Graphs and homomorphisms*, volume 28 of *Oxford lecture series in mathematics and its applications*. Oxford University Press, 2004.
- 18 Dan Kalman. The generalized Vandermonde matrix. *Mathematics Magazine*, 57(1):15–21, 1984. doi:10.2307/2690290.
- 19 Grigoris Karvounarakis and Todd J. Green. Semiring-annotated data: queries and provenance? *SIGMOD Rec.*, 41(3):5–14, 2012. doi:10.1145/2380776.2380778.
- 20 Mahmoud Abo Khamis, Hung Q. Ngo, Reinhard Pichler, Dan Suciu, and Yisu Remy Wang. Convergence of datalog over (pre-) semirings. In *PODS '22: Int. Conf. on Management of Data, Philadelphia, 2022*, pages 105–117. ACM, 2022. doi:10.1145/3517804.3524140.
- 21 Egor V. Kostylev, Juan L. Reutter, and András Z. Salamon. Classification of annotation semirings over containment of conjunctive queries. *ACM Trans. Database Syst.*, 39(1):1:1–1:39, 2014. doi:10.1145/2556524.
- 22 Gábor Kun. Constraints, MMSNP and expander relational structures. *Combinatorica*, 33(3):335–347, 2013. doi:10.1007/s00493-013-2405-4.
- 23 Gábor Kun and Mario Szegedy. A new line of attack on the dichotomy conjecture. *European Journal of Combinatorics*, 52:338–367, 2016. Special Issue: Recent Advances in Graphs and Analysis. doi:10.1016/j.ejc.2015.07.011.
- 24 Jaroslaw Kwiecien, Jerzy Marcinkowski, and Piotr Ostropolski-Nalewaja. Determinacy of real conjunctive queries. the boolean case. In *Proceedings of the 41st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS '22*, pages 347–358, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3517804.3524168.
- 25 László Lovász. Operations with structures. *Acta Math. Acad. Sci. Hungar*, 18(3-4):321–328, 1967.
- 26 Alan Nash, Luc Segoufin, and Victor Vianu. Views and queries: Determinacy and rewriting. *ACM Trans. Database Syst.*, 35(3), jul 2010. doi:10.1145/1806907.1806913.
- 27 Benjamin Rossman. Homomorphism preservation theorems. *Journal of the ACM (JACM)*, 55(3):1–53, 2008. doi:10.1145/1379759.1379763.
- 28 Claude Tardif, Cynthia Loten, and Benoit Larose. A characterisation of first-order constraint satisfaction problems. *Logical Methods in Computer Science*, 3(4), 2007. doi:10.2168/LMCS-3(4:6)2007.
- 29 Dmitriy Zhuk. A proof of CSP dichotomy conjecture. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 331–342. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.38.

# Approximating Single-Source Personalized PageRank with Absolute Error Guarantees

Zhewei Wei<sup>1</sup>  

Renmin University of China, Beijing, China

Ji-Rong Wen  

Renmin University of China, Beijing, China

Mingji Yang  

Renmin University of China, Beijing, China

---

## Abstract

*Personalized PageRank (PPR)* is an extensively studied and applied node proximity measure in graphs. For a pair of nodes  $s$  and  $t$  on a graph  $G = (V, E)$ , the PPR value  $\pi(s, t)$  is defined as the probability that an  $\alpha$ -discounted random walk from  $s$  terminates at  $t$ , where the walk terminates with probability  $\alpha$  at each step. We study the classic *Single-Source PPR query*, which asks for PPR approximations from a given source node  $s$  to all nodes in the graph. Specifically, we aim to provide approximations with *absolute error* guarantees, ensuring that the resultant PPR estimates  $\hat{\pi}(s, t)$  satisfy  $\max_{t \in V} |\hat{\pi}(s, t) - \pi(s, t)| \leq \varepsilon$  for a given error bound  $\varepsilon$ . We propose an algorithm that achieves this with high probability, with an expected running time of

- $\tilde{O}(\sqrt{m}/\varepsilon)$  for directed graphs<sup>2</sup>, where  $m = |E|$ ;
- $\tilde{O}(\sqrt{d_{\max}}/\varepsilon)$  for undirected graphs, where  $d_{\max}$  is the maximum node degree in the graph;
- $\tilde{O}(n^{\gamma-1/2}/\varepsilon)$  for power-law graphs, where  $n = |V|$  and  $\gamma \in (\frac{1}{2}, 1)$  is the extent of the power law.

These sublinear bounds improve upon existing results. We also study the case when *degree-normalized absolute error* guarantees are desired, requiring  $\max_{t \in V} |\hat{\pi}(s, t)/d(t) - \pi(s, t)/d(t)| \leq \varepsilon_d$  for a given error bound  $\varepsilon_d$ , where the graph is undirected and  $d(t)$  is the degree of node  $t$ . We give an algorithm that provides this error guarantee with high probability, achieving an expected complexity of  $\tilde{O}(\sqrt{\sum_{t \in V} \pi(s, t)/d(t)}/\varepsilon_d)$ . This improves over the previously known  $O(1/\varepsilon_d)$  complexity.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Graph algorithms analysis; Theory of computation  $\rightarrow$  Streaming, sublinear and near linear time algorithms

**Keywords and phrases** Graph Algorithms, Sublinear Algorithms, Personalized PageRank

**Digital Object Identifier** 10.4230/LIPIcs.ICDT.2024.9

**Related Version** *Full Version*: <https://arxiv.org/pdf/2401.01019.pdf> [50]

**Funding** This work was partially done at Gaoling School of Artificial Intelligence, Peng Cheng Laboratory, Beijing Key Laboratory of Big Data Management and Analysis Methods, and MOE Key Lab of Data Engineering and Knowledge Engineering. This research was supported in part by National Natural Science Foundation of China (No. U2241212, No. 61972401, No. 61932001, No. 61832017, No. U2001212), the major key project of PCL (PCL2021A12), Beijing Natural Science Foundation (No. 4222028), Beijing Outstanding Young Scientist Program No.BJJWZYJH012019100020098, Alibaba Group through Alibaba Innovative Research Program, and Huawei-Renmin University joint program on Information Retrieval.

**Acknowledgements** We also wish to acknowledge the support provided by Engineering Research Center of Next-Generation Intelligent Search and Recommendation, Ministry of Education. Finally, we thank the anonymous reviewers for their valuable comments.

---

<sup>1</sup> Zhewei Wei is the corresponding author.

<sup>2</sup>  $\tilde{O}(\cdot)$  suppresses polylog( $n$ ) factors.



## 1 Introduction

In graph mining, computing *node proximity* values efficiently is a fundamental problem with broad applications, as they provide quantitative amounts to measure the closeness or relatedness between the nodes. A basic and extensively used proximity measure is *Personalized PageRank (PPR)* [14], which is a direct variant of Google’s renowned *PageRank* centrality [14]. PPR has found multifaced applications for local graph partitioning [4, 53, 17], node embedding [37, 40, 54], and graph neural networks [27, 12, 42], among many others [21].

We study the classic problem of approximating *Single-Source PPR (SSPPR)*, where we are given a source node  $s$  in the graph, and our goal is to approximate the PPR values of all nodes in the graph w.r.t.  $s$ . Particularly, we concentrate on the complexity bounds for approximating SSPPR with *absolute error* or *degree-normalized absolute error* guarantees. After examining the existing bounds for the problem, we present novel algorithms with improved complexities to narrow the margin between the previous upper bounds and the known lower bounds.

In the remainder of this section, we formally state the problem, discuss the existing bounds, and introduce our motivations and contributions.

### 1.1 Problem Formulation

We consider a directed or undirected graph  $G = (V, E)$ , where  $|V| = n$  and  $|E| = m$ . For undirected graphs, we conceptually view each undirected edge as two opposing directed edges. We assume that every node in  $V$  has a nonzero out-degree.

A *random walk* on  $G$  starts from some *source node*  $s \in V$  and, at each step, transitions to an out-neighbor of the current node chosen uniformly at random. For a constant *decay factor*  $\alpha \in (0, 1)$ , an  $\alpha$ -*discounted random walk* proceeds in the same way as a random walk, except that it terminates with probability  $\alpha$  before each step. The *Personalized PageRank (PPR)* value for a pair of nodes  $s$  and  $t$  in  $V$ , denoted by  $\pi(s, t)$ , is defined as the probability that an  $\alpha$ -discounted random walk from  $s$  terminates at  $t$ .

We study the problem of estimating *Single-Source PPR (SSPPR)*, that is, deriving  $\hat{\pi}(s, t)$  as estimates for PPR values  $\pi(s, t)$  from a given source node  $s$  to all  $t \in V$ . We focus on the complexities of approximating SSPPR with *absolute error* or *degree-normalized absolute error* guarantees. The corresponding two types of queries, dubbed as the *SSPPR-A query* and the *SSPPR-D query*, are formally defined below.

► **Definition 1** (SSPPR-A Query: Approximate SSPPR Query with Absolute Error Bounds). *Given a source node  $s \in V$  and an error parameter  $\varepsilon$ , the query requires PPR estimates  $\hat{\pi}(s, t)$  for all  $t \in V$ , such that  $|\hat{\pi}(s, t) - \pi(s, t)| \leq \varepsilon$  holds for all  $t \in V$ .*

► **Definition 2** (SSPPR-D Query: Approximate SSPPR Query with Degree-Normalized Absolute Error Bounds). *On an undirected graph, given a source node  $s \in V$  and an error parameter  $\varepsilon_d$ , the query requires PPR estimates  $\hat{\pi}(s, t)$  for all  $t \in V$ , such that  $|\hat{\pi}(s, t)/d(t) - \pi(s, t)/d(t)| \leq \varepsilon_d$  holds for all  $t \in V$ . Here,  $d(t)$  denotes the degree of node  $t$ .*

In a word, the SSPPR-A query requires the maximum absolute error to be bounded above by  $\varepsilon$ , while the SSPPR-D query considers the absolute errors normalized (i.e., divided) by the degree of each node. Note that we restrict the SSPPR-D query to undirected graphs.

This paper aims to develop *sublinear* algorithms for the SSPPR-A and SSPPR-D queries with improved complexities over existing methods. By “sublinear algorithms,” we refer to algorithms whose complexity bounds are sublinear in the size of the graph (but they can

simultaneously depend on the error parameter, e.g.,  $O(\sqrt{n}/\varepsilon)$  is considered sublinear). We allow the algorithms to return the results as a sparse vector, which enables the output size to be  $o(n)$ . Also, we regard the algorithms as acceptable if they answer the queries *with high probability (w.h.p.)*, defined as “with probability at least  $1 - 1/n$ .”

## 1.2 Prior Complexity Bounds

### Lower Bounds

To our knowledge, only trivial lower bounds are known for the SSPPR-A and SSPPR-D queries. More precisely, for the SSPPR-A query, since the algorithm needs to return nonzero estimates for those nodes  $t$  with  $\pi(s, t) > \varepsilon$ , and there may exist  $\Theta(\min(1/\varepsilon, n))$  such nodes (note that  $\sum_{t \in V} \pi(s, t) = 1$ ), the trivial lower bound for answering the query is  $\Omega(\min(1/\varepsilon, n))$ . As we are considering sublinear bounds, we write this as  $\Omega(1/\varepsilon)$ .

Similarly, for the SSPPR-D query, the algorithm needs to return nonzero estimates for nodes  $t$  with  $\pi(s, t)/d(t) > \varepsilon_d$ , so the lower bound is  $\Omega(1/\varepsilon_d \cdot \sum_{t \in V} \pi(s, t)/d(t))$ . Note that  $\sum_{t \in V} \pi(s, t)/d(t) < \sum_{t \in V} \pi(s, t) = 1$  for nontrivial graphs. Additionally, in the full version of this paper [50], we show that if each source node  $s \in V$  is chosen with probability proportional to its degree (i.e., with probability  $d(s)/(2m)$ ), the average lower bound becomes  $\Omega(1/\varepsilon_d \cdot n/m)$ .

### Upper Bounds

From a theoretical point of view, we summarize the best bounds to date for the SSPPR-A query as follows.

- The *Monte Carlo method* [16] straightforwardly simulates a number of  $\alpha$ -discounted random walks from  $s$ , and computes the fraction of random walks that terminate at  $t$  as the estimate  $\hat{\pi}(s, t)$  for each  $t \in V$ . By standard Chernoff bound arguments, it requires expected  $\tilde{O}(1/\varepsilon^2)$  time to achieve the guarantees w.h.p.
- *Forward Push* [4, 5] is a celebrated “local-push” algorithm for the SSPPR query, which takes as input a parameter  $r_{\max}$  and runs in  $O(1/r_{\max})$  time. However, it only guarantees that the degree-normalized absolute error (i.e.,  $\max_{v \in V} |\hat{\pi}(s, v)/d(v) - \pi(s, v)/d(v)|$ ) is bounded by  $r_{\max}$  on undirected graphs, and that the  $\ell_1$ -error (i.e.,  $\sum_{v \in V} |\hat{\pi}(s, v) - \pi(s, v)|$ ) is bounded by  $m \cdot r_{\max}$  on directed graphs. Thus, if we apply Forward Push to answer the SSPPR-A query: on undirected graphs, we need to set  $r_{\max} = \varepsilon/d_{\max}$ , where  $d_{\max}$  is the maximum node degree in  $G$ ; on directed graphs, we need to set  $r_{\max} = \varepsilon/m$ . These settings lead to pessimistic bounds of  $O(d_{\max}/\varepsilon)$  and  $O(m/\varepsilon)$ , respectively. Note that  $d_{\max}$  can reach  $\Theta(n)$  in the worst case, and the  $O(m/\varepsilon)$  bound is not sublinear.
- *Backward Push* [1, 2] is a “local-push” algorithm for approximating  $\pi(v, t)$  from all  $v \in V$  to a given *target node*  $t \in V$ , known as the *Single-Target PPR (STPPR)* query. It takes as input a parameter  $r_{\max}$  and cleanly returns estimates with an absolute error bound of  $r_{\max}$ . However, if we enforce Backward Push to answer the SSPPR-A query, we need to perform it with  $r_{\max} = \varepsilon$  for each  $t \in V$ , resulting in a complexity of  $O(m/\varepsilon)$  again. This bound is inferior, but we mention it here since it enlightens our algorithms.

In conclusion, the currently best sublinear bounds for the SSPPR-A query are  $\tilde{O}(1/\varepsilon^2)$  provided by Monte Carlo and  $O(d_{\max}/\varepsilon)$  on undirected graphs by Forward Push.

As for the SSPPR-D query, Forward Push [5] provides an elegant  $O(1/\varepsilon_d)$  bound. To our knowledge, no other prior methods are explicitly tailored to the SSPPR-D query.

### 1.3 Motivations

#### Motivations for the SSPPR-A Query

Although approximating SSPPR with absolute error guarantees is a natural problem, surprisingly, it has not been studied in depth in the literature. We believe that this is partly because of its inherent hardness. In particular, a line of recent research for approximating SSPPR [46, 30, 51, 29, 28] mainly focuses on providing *relative error* guarantees for PPR values above a specified threshold. We note that absolute error guarantees are harder to achieve than relative or degree-normalized absolute error guarantees, as the latter ones allow larger actual errors for nodes with larger PPR values or degrees. Specifically, an SSPPR algorithm with absolute error guarantees can be directly modified to obtain relative or degree-normalized absolute error guarantees.

In contrast, an interesting fact is that, for the relatively less-studied STPPR query, a simple Backward Push is sufficient and efficient for absolute error guarantees. As a result, when PPR values with absolute error guarantees are desired in some applications, STPPR methods are employed instead of SSPPR methods [54, 43].

These facts stimulate us to derive better bounds for the SSPPR-A query. As discussed, a large gap exists between the existing upper bounds and the lower bound of  $\Omega(1/\varepsilon)$ . The previous upper bounds, namely  $\tilde{O}(1/\varepsilon^2)$ ,  $O(m/\varepsilon)$ , and  $O(d_{\max}/\varepsilon)$  on undirected graphs, motivate us to devise a new algorithm that:

- runs in linear time w.r.t.  $1/\varepsilon$ ;
- runs in sublinear time w.r.t.  $m$ ;
- beats the  $O(d_{\max}/\varepsilon)$  bound on undirected graphs.

#### Motivations for the SSPPR-D Query

Our study of the SSPPR-D query is motivated by a classic approach of using approximate SSPPR to perform *local graph partitioning* [5, 3, 53, 17]. This task aims to detect a cut with provably small conductance near a specified seed node without scanning the whole graph. To this end, this classic approach computes approximate PPR values  $\hat{\pi}(s, v)$  from the seed node  $s$ , sorts the nodes in decreasing order of  $\hat{\pi}(s, v)/d(v)$ , and then finds a desired cut based on this order. As the quality of this approach relies heavily on the approximation errors of the values  $\hat{\pi}(s, v)/d(v)$ , it is natural to consider the SSPPR-D query. Notably, in carrying out this framework, the seminal and celebrated **PageRank-Nibble** algorithm [5] employs Forward Push as a subroutine for approximating PPR values. As it turns out, the error bounds of Forward Push match the requirements of the SSPPR-D query, and its cost dominates the overall complexity of **PageRank-Nibble**. Therefore, an improved upper bound for the SSPPR-D query can potentially lead to faster local graph partitioning algorithms.

However, the SSPPR-D query is rarely studied afterward despite its significance. To our knowledge, no existing method overcomes the  $O(1/\varepsilon_d)$  bound of Forward Push, nor has any previous work pointed out the gap between this bound and the aforementioned lower bound. Motivated by this, we formulate this problem and propose an algorithm that beats the known  $O(1/\varepsilon_d)$  bound.

### 1.4 Our Results

We propose algorithms for the SSPPR-A and SSPPR-D queries under a unified framework, melding Monte Carlo and Backward Push in a novel and nontrivial way. Roughly speaking, we use Backward Push to reduce the variances of the Monte Carlo estimators, and we propose



a novel technique called *Adaptive Backward Push* to control the cost of Backward Push for each node and balance its total cost with that of Monte Carlo. We summarize the improved bounds achieved by our algorithms as follows.

### Improved Upper Bounds for the SSPPR-A Query

We present an algorithm that answers the SSPPR-A query w.h.p., with a complexity of:

- expected  $\tilde{O}(\sqrt{m}/\varepsilon)$  for directed graphs;
- expected  $\tilde{O}(\sqrt{d_{\max}}/\varepsilon)$  for undirected graphs.

These bounds are strictly sublinear in  $m$  and linear in  $1/\varepsilon$ . Also, the  $\tilde{O}(\sqrt{d_{\max}}/\varepsilon)$  bound improves over the previous  $O(d_{\max}/\varepsilon)$  bound by up to a factor of  $\Theta(\sqrt{n})$ .

Additionally, we study the special case that the underlying graph is a *power-law graph* (a.k.a. *scale-free graph*). This is a renowned and widely used model for describing large real-world graphs [10, 13]. Under power-law assumptions (see Assumption 3 in Subsection 3.3), we prove that the complexity of our algorithm diminishes to  $\tilde{O}(n^{\gamma-1/2}/\varepsilon)$  for both directed and undirected graphs, where  $\gamma \in (\frac{1}{2}, 1)$  is the extent of the power law. Notably, as  $\gamma < 1$ , we have  $\gamma - \frac{1}{2} < \frac{1}{2}$ , so this bound is strictly  $o(\sqrt{n}/\varepsilon)$ . Also, when  $\gamma \rightarrow \frac{1}{2}$ , this bound approaches  $\tilde{O}(1/\varepsilon)$ , matching the lower bound of  $\Theta(1/\varepsilon)$  up to logarithmic factors. We summarize the complexity bounds of answering the SSPPR-A query in Table 1.

► **Remark.** Our algorithm for the SSPPR-A query can be adapted to approximate a more generalized form of Personalized PageRank [14], where the source node is randomly chosen from a given probability distribution vector. We only need to construct an alias structure [41] for the distribution (this can be done in asymptotically the same time as inputting the vector) so that we can sample a source node in  $O(1)$  time when performing Monte Carlo. This modification does not change our algorithm's error guarantees and complexity bounds. Particularly, this allows us to estimate the PageRank [14] values, in which case we can sample the source nodes uniformly at random from  $V$  without using the alias method.

### Improved Upper Bounds for the SSPPR-D Query

We present an algorithm that answers the SSPPR-D query w.h.p., with an expected complexity of  $\tilde{O}\left(1/\varepsilon_d \cdot \sqrt{\sum_{t \in V} \pi(s, t)/d(t)}\right)$ . This improves upon the previous  $O(1/\varepsilon_d)$  bound of Forward Push towards the lower bound of  $\Omega\left(1/\varepsilon_d \cdot \sum_{t \in V} \pi(s, t)/d(t)\right)$ . To see the superiority of our bound, let us consider the case when each node  $s \in V$  is chosen as the source node with probability  $d(s)/(2m)$ . This setting corresponds to the practical scenario where a node with larger importance is more likely to be chosen as the source node. We show that under this setting, our bound becomes  $\tilde{O}\left(1/\varepsilon_d \cdot \sqrt{n/m}\right)$ , which is lower than  $O(1/\varepsilon_d)$  by up to a factor of  $\Theta(\sqrt{n})$ . Recall that under this setting, the lower bound becomes  $\Omega(1/\varepsilon_d \cdot n/m)$ . In Table 2, we summarize the complexity bounds of answering the SSPPR-D query.

► **Remark.** If we treat  $\alpha$  as a variable (as is the case in the context of local graph partitioning), the complexity bounds of our algorithms for these two queries both exhibit a linear dependence on  $1/\alpha$ , which is the same as existing upper bounds. For the sake of simplicity, we treat  $\alpha$  as a constant and omit this term in this work.

**Paper Organization.** The remainder of this paper is organized as follows. Section 2 discusses some related work for PPR computation, and Section 3 offers the preliminaries. Section 4 presents the ideas and the main procedure of our proposed algorithm for the SSPPR-A query. In Section 5, we prove our results for the SSPPR-A query by analyzing our proposed algorithm. Some deferred proofs and our algorithm and analyses for the SSPPR-D query can be found in the full version of this paper [50].



■ **Table 1** Complexity bounds of answering the SSPPR-A query on different types of graphs. For power-law graphs, the graph can be either directed or undirected, and  $\gamma \in (\frac{1}{2}, 1)$  denotes the exponent of the power law. We plug in  $m = \tilde{O}(n)$  for power-law graphs.

	Directed Graphs	Undirected Graphs	Power-Law Graphs
Monte Carlo [16]	$\tilde{O}\left(\frac{1}{\varepsilon^2}\right)$	$\tilde{O}\left(\frac{1}{\varepsilon^2}\right)$	$\tilde{O}\left(\frac{1}{\varepsilon^2}\right)$
Forward Push [4]	$O\left(\frac{m}{\varepsilon}\right)$	$O\left(\frac{d_{\max}}{\varepsilon}\right)$	$\tilde{O}\left(\frac{n}{\varepsilon}\right)$
Ours	$\tilde{O}\left(\frac{\sqrt{m}}{\varepsilon}\right)$	$\tilde{O}\left(\frac{\sqrt{d_{\max}}}{\varepsilon}\right)$	$\tilde{O}\left(\frac{n^{\gamma-1/2}}{\varepsilon}\right) = o\left(\frac{\sqrt{n}}{\varepsilon}\right)$ , approaching $\tilde{O}\left(\frac{1}{\varepsilon}\right)$ when $\gamma \rightarrow \frac{1}{2}$

■ **Table 2** Complexity bounds of answering the SSPPR-D query on undirected graphs.

	Parameterized complexity for a given $s$	Average complexity when each $s \in V$ is chosen with probability $d(s)/(2m)$
Forward Push [4]	$O\left(\frac{1}{\varepsilon_d}\right)$	$O\left(\frac{1}{\varepsilon_d}\right)$
Lower Bound	$\Omega\left(\frac{1}{\varepsilon_d} \sum_{t \in V} \frac{\pi(s, t)}{d(t)}\right)$	$\Omega\left(\frac{1}{\varepsilon_d} \cdot \frac{n}{m}\right)$
Ours	$\tilde{O}\left(\frac{1}{\varepsilon_d} \sqrt{\sum_{t \in V} \frac{\pi(s, t)}{d(t)}}\right)$	$\tilde{O}\left(\frac{1}{\varepsilon_d} \sqrt{\frac{n}{m}}\right)$

## 2 Other Related Work

As a classic task in graph mining, PPR computation has been extensively studied in the past decades, and numerous efficient approaches have been proposed. Many recent methods combine the basic techniques of Monte Carlo, Forward Push, and Backward Push to achieve improved efficiency [32, 45, 49, 46, 30, 51, 29, 28]. A key ingredient in integrating these techniques is the *invariant* equation provided by Forward Push or Backward Push. While our algorithms also leverage the invariant of Backward Push to unify it with Monte Carlo, we adopt a novel approach based on Adaptive Backward Push and conduct different analyses.

For SSPPR approximation, FORA [47, 46] is a representative sublinear algorithm among a recent line of research [46, 30, 51, 29, 28]. FORA uses Forward Push and Monte Carlo to provide relative error guarantees for PPR values above a specified threshold w.h.p., and the subsequent work proposes numerous optimizations for it. However, this method cannot be directly applied to the SSPPR-A query. A notable extension of FORA is SpeedPPR [51], which further incorporates Power Method [14] to achieve higher efficiency. Nevertheless, the complexity of SpeedPPR is no longer sublinear. Among other studies for SSPPR [11, 59, 35, 39, 15, 26, 56], BEAR [39] and BEPI [26] are two representative approaches based on matrix manipulation. However, they incur inferior complexities due to the large overhead of matrix computation.

---

**Algorithm 1** BackwardPush.
 

---

**Input:** graph  $G$ , decay factor  $\alpha$ , target node  $t$ , threshold  $r_{\max}$   
**Output:** backward reserves  $q(v, t)$  and residues  $r(v, t)$  for all  $v \in V$

- 1  $q(v, t) \leftarrow 0$  for all  $v \in V$
- 2  $r(t, t) \leftarrow 1$  and  $r(v, t) \leftarrow 0$  for all  $v \in V \setminus \{t\}$
- 3 **while**  $\exists v \in V$  such that  $r(v, t) > r_{\max}$  **do**
- 4     pick an arbitrary  $v \in V$  with  $r(v, t) > r_{\max}$
- 5     **for each**  $u \in \mathcal{N}_{\text{in}}(v)$  **do**
- 6          $r(u, t) \leftarrow r(u, t) + (1 - \alpha) \cdot r(v, t) / d_{\text{out}}(u)$
- 7      $q(v, t) \leftarrow q(v, t) + \alpha \cdot r(v, t)$
- 8      $r(v, t) \leftarrow 0$
- 9 **return**  $q(v, t)$  and  $r(v, t)$  for all  $v \in V$

---

There also exist many studies for other PPR queries, such as Single-Pair query [20, 33, 32, 45], Single-Target query [43], and top- $k$  query [7, 18, 20, 19, 52, 49]. Some recent work further considers computing PPR on dynamic graphs [57, 36, 58, 55] or in parallel/distributed settings [8, 22, 23, 38, 31, 44, 24]. These methods often utilize specifically designed methodologies and techniques, hence they are orthogonal to our work.

To sum up, despite the large body of studies devoted to PPR computation, the SSPPR-A and SSPPR-D queries are still not explored in depth. This is because the relevant approaches either are unsuitable for these two queries or exhibit at least linear complexities. We also note that many related studies optimize PPR computation from an engineering viewpoint instead of a theoretical one, and thus they do not provide better complexity bounds.

### 3 Notations and Tools

#### 3.1 Notations

We use  $d_{\text{in}}(v)$  and  $d_{\text{out}}(v)$  to denote the in-degree and out-degree of a node  $v \in V$ , respectively. Additionally,  $\mathcal{N}_{\text{in}}(v)$  and  $\mathcal{N}_{\text{out}}(v)$  denote the in-neighbor and out-neighbor set of  $v$ , respectively. In the case of undirected graphs, we use  $d(v)$  to represent the degree of  $v$ , and we define  $d_{\max} = \max_{v \in V} d(v)$  to indicate the maximum degree in  $G$ .

#### 3.2 Backward Push

Backward Push [2, 34] is a simple and classic algorithm for approximating STPPR, that is, estimating  $\pi(v, t)$  from all  $v \in V$  to a given target node  $t \in V$ . It works by repeatedly performing *reverse pushes*, which conceptually simulate random walks from the backward direction deterministically. It takes as input a parameter  $r_{\max}$  to control the depth of performing the pushes: a smaller  $r_{\max}$  leads to deeper pushes.

Specifically, Backward Push maintains *reserves*  $q(v, t)$  and *residues*  $r(v, t)$  for all  $v \in V$ , where  $q(v, t)$  is an underestimate of  $\pi(v, t)$  and  $r(v, t)$  is the probability mass to be propagated. A reverse push operation for a node  $v$  transfers  $\alpha$  portion of  $r(v, t)$  to  $q(v, t)$  and propagates the remaining probability mass to the in-neighbors of  $v$ , as per the probability that a random walk at the in-neighbors proceeds to  $v$ . As shown in Algorithm 1, Backward Push initially sets all reserves and residues to be 0 except that  $r(t, t) = 1$ , and then repeatedly performs reverse pushes to nodes  $v$  with  $r(v, t) > r_{\max}$ . After that, it returns  $q(v, t)$ 's as the estimates.

In this paper, we use the following properties of Backward Push [34]:

- The results of Backward Push satisfy  $r(v, t) \leq r_{\max}$  and  $|q(v, t) - \pi(v, t)| \leq r_{\max}$ ,  $\forall v \in V$ .
- The results of Backward Push satisfy the following invariant:

$$\pi(v, t) = q(v, t) + \sum_{u \in V} \pi(v, u)r(u, t), \quad \forall v \in V. \quad (1)$$

- The complexity of Backward Push is

$$O\left(\frac{1}{r_{\max}} \sum_{v \in V} \pi(v, t)d_{\text{in}}(v)\right). \quad (2)$$

- Running Backward Push with parameter  $r_{\max}$  for each  $t \in V$  takes  $O(m/r_{\max})$  time.

### 3.3 Power-Law Assumption

Power-law graphs are an extensively used model for describing real-world graphs [10, 13]. Regarding PPR computation, it is observed in [9] that the PPR values on power-law graphs also follow a power-law distribution. Formally, in our analyses for power-law graphs, we use the following assumption, which has been adopted in several relevant works for graph analysis [9, 32, 48]:

► **Assumption 3 (Power-Law Graph).** *In a power-law graph, for any source node  $v \in V$ , the  $i$ -th largest PPR value w.r.t.  $v$  equals  $\Theta\left(\frac{i^{-\gamma}}{n^{1-\gamma}}\right)$ , where  $1 \leq i \leq n$  and  $\gamma \in (\frac{1}{2}, 1)$  is the exponent of the power law.*

## 4 Our Algorithm for the SSPPR-A Query

This section presents our algorithm for answering the SSPPR-A query with improved complexity bounds. Our algorithm for the SSPPR-D query is given in the full version of this paper [50]. Before diving into the details, we give high-level ideas and introduce key techniques for devising and analyzing the algorithm.

### 4.1 High-Level Ideas

Recall that for the SSPPR-A query, a fixed absolute error bound is required for each node  $t \in V$ . We find it hard to achieve this using Forward Push and Monte Carlo, as they inherently incur larger errors for nodes with larger degrees or PPR values (for Monte Carlo, this can be seen when analyzing it using Chernoff bounds). Thus, to answer the SSPPR-A query, it is crucial to reduce the errors for these hard-case nodes efficiently. A straightforward idea is to run Backward Push from these nodes, although using an STPPR algorithm to answer the SSPPR query seems counterintuitive. As performing Backward Push alone for all nodes requires  $O(m/\varepsilon)$  time, we combine it with Monte Carlo to achieve a lower complexity.

In a word, our proposed algorithms employ Backward Push to reduce the number of random walks needed in Monte Carlo. Intuitively, by running Backward Push for a node  $t$ , we simulate random walks backward from  $t$ . Consequently, when performing forward random walks in Monte Carlo, our objective shifts from reaching node  $t$  to reaching the intermediate nodes touched by Backward Push. This method significantly reduces the variances of the Monte Carlo estimators since the random walks can increment the estimated values even if they fail to reach  $t$ . However, a major difficulty is that we cannot afford to perform deep Backward Push for each  $t \in V$ , which is both expensive and unnecessary. To address this issue, we propose the following central technique: Adaptive Backward Push.

**Adaptive Backward Push.** A crucial insight behind our algorithms is that performing deep Backward Push for each  $t \in V$  is wasteful. This is because doing so yields accurate estimates for  $\pi(v, t)$  for all  $v \in V$ , but for SSPPR queries, only  $\pi(s, t)$  is required to be estimated. Thus, instead of performing Backward Push deeply to propagate enough probability mass to each node, we only need to ensure that the probability mass pushed backward from  $t$  to  $s$  is sufficient to yield an accurate estimate for  $\pi(s, t)$ . This motivates us to perform Backward Push *adaptively*. More precisely, we wish to set smaller  $r_{\max}(t)$  for  $t$  with larger  $\pi(s, t)$ , rendering the Backward Push process for them deeper. An intuitive explanation is that, for larger  $\pi(s, t)$ , the minimally acceptable estimates  $\pi(s, t) - \varepsilon$  are larger, so we need to perform Backward Push deeper to push more probability mass to  $s$ . As an extreme example, if we know that  $\pi(s, t) \leq \varepsilon$  for some  $t$ , then we can simply return  $\hat{\pi}(s, t) = 0$  as its PPR approximation, so we do not need to perform Backward Push for these nodes with small PPR values. On the other hand, our technique also adaptively balances the cost of Backward Push and Monte Carlo, which will be introduced in the next subsection.

To implement Adaptive Backward Push, a natural idea would be directly setting  $r_{\max}(t)$  to be inversely proportional to  $\pi(s, t)$ . At first glance, this idea seems paradoxical since the  $\pi(s, t)$  values are exactly what we aim to estimate. However, it turns out that rough estimates of them suffice for our purpose. In fact, Monte Carlo offers a simple and elegant way to roughly approximate  $\pi(s, t)$  with relatively low overheads. Thus, we need to run Monte Carlo to obtain rough PPR estimates before performing Adaptive Backward Push. Despite the simplicity of the ideas, the detailed procedure of the algorithm and its analyses are nontrivial, as we elaborate below.

## 4.2 Techniques

Now, we introduce several additional techniques used in our algorithms.

**Three-Phase Framework.** As discussed above, before performing Adaptive Backward Push, we need to perform Monte Carlo to obtain rough PPR estimates. Also, the results of Backward Push and Monte Carlo are combined to derive the final results. For ease of analysis, we conduct Monte Carlo twice, yielding two independent sets of approximations. This leads to our *three-phase framework*:

- (I) running Monte Carlo to obtain rough PPR estimates;
- (II) performing Adaptive Backward Push to obtain backward reserves and residues;
- (III) running Monte Carlo and combining the results with those of Backward Push to yield the final estimates.

However, there are some difficulties in carrying out this framework. We discuss them in detail below and provide a more accurate description of our algorithm in Subsection 4.3.

**Identifying Candidate Nodes.** As mentioned earlier, if we know that  $\pi(s, t) \leq \varepsilon$  for some  $t$ , then we can safely return  $\hat{\pi}(s, t) = 0$ . This means that we only need to consider nodes  $t$  with  $\pi(s, t) > \varepsilon$ . Fortunately, when running Monte Carlo to obtain rough estimates, we can also identify these nodes (w.h.p.), thus avoiding the unnecessary cost of performing Adaptive Backward Push for other nodes. We use  $C$  to denote the *candidate set* that consists of these identified *candidate nodes*. In light of this, Phase I serves two purposes: determining the candidate nodes  $t \in C$  and obtaining rough estimates for their PPR values, denoted as  $\pi'(s, t)$ 's. Subsequently, in Phase II, we perform Adaptive Backward Push for these candidate nodes  $t \in C$ , where the thresholds are set to be inversely proportional to  $\pi'(s, t)$ .

**Estimation Formula.** Let us take a closer look at the results of Backward Push for a candidate node  $t \in C$ . From the invariant of Backward Push (Equation 1), we have

$$\pi(s, t) = q(s, t) + \sum_{v \in V} \pi(s, v)r(v, t).$$

This essentially expresses the desired PPR value  $\pi(s, t)$  as  $q(s, t)$  plus a linear combination of  $\pi(s, v)$ 's with coefficients  $r(v, t)$ 's. Now, we can obtain an estimate  $\hat{\pi}(s, t)$  by substituting  $\pi(s, v)$ 's by their Monte Carlo estimates obtained in Phase III, denoted by  $\pi''(s, v)$ 's:

$$\hat{\pi}(s, t) = q(s, t) + \sum_{v \in V} \pi''(s, v)r(v, t). \quad (3)$$

As Backward Push guarantees that  $r(v, t) \leq r_{\max}$  for all  $v \in V$ , the coefficients of these Monte Carlo estimates are small, making the variance of  $\hat{\pi}(s, t)$  small. Thus, by carefully setting the parameters of Backward Push and Monte Carlo, we can bound this variance and apply *Chebyshev's inequality* to obtain the desired absolute error bound on  $\hat{\pi}(s, t)$  with constant success probability. Then, we leverage the *median trick* [25] (see Appendix A) to amplify this probability while only introducing an additional logarithmic factor to the complexity. Therefore, in Phase III, we run Monte Carlo several times to obtain independent samples of  $\hat{\pi}(s, t)$ , denoted as  $\hat{\pi}_i(s, t)$ . After that, we compute their median values as the final estimates.

**Balancing Phases II and III.** Finally, to optimize the overall complexity of the algorithm, it is essential to strike a balance between the cost of Phases II and III. In fact, the following balancing strategy is another manifestation of the adaptiveness of our algorithm. In particular, performing more Adaptive Backward Push in Phase II results in fewer random walks needed in Phase III. In our analysis part (Section 5), we find that the complexity bound of Adaptive Backward Push is inversely proportional to the number of random walk samplings in Phase III (denoted as  $n_r$ ), and the cost of Monte Carlo in Phase III is proportional to  $n_r$ . The problem is that we do not know a priori the optimal setting of  $n_r$  in terms of balancing Phases II and III. As a workaround, we propose to try running Adaptive Backward Push with exponentially decreasing  $n_r$ , and terminate this process once its paid cost exceeds the expected cost of simulating  $n_r$  random walks. Intuitively, in this way, our algorithm achieves the same asymptotic complexity as if it knew the optimal  $n_r$ . We will describe the detailed process and formally state this claim below.

### 4.3 Main Algorithm

Algorithm 2 outlines the pseudocode of our proposed algorithm for the SSPPR-A query. It consists of three phases: Phase I (line 2 to line 3), Phase II (line 5 to line 14), and Phase III (line 16 to line 19). We recap the purposes of the three phases as follows:

- (I) running Monte Carlo to obtain estimates  $\pi'(s, v)$  and derive the candidate set  $C$ ;
- (II) performing Adaptive Backward Push to obtain reserves  $q(s, t)$  and residues  $r(v, t)$  for candidate nodes  $t \in C$ , where the associated  $n_r$  (the number of random walk samplings in Phase III) is exponentially decreased and the stopping rule is designed to balance Phases II and III;
- (III) running Monte Carlo several times, combining their results  $\pi''(s, v)$  with those of Adaptive Backward Push, and finally taking the median values as the results  $\hat{\pi}(s, t)$ .

■ **Algorithm 2** Our algorithm for the SSPPR-A query.

---

**Input:** graph  $G = (V, E)$ , decay factor  $\alpha$ , source node  $s$ , error parameter  $\varepsilon$   
**Output:** estimates  $\hat{\pi}(s, t)$  for all  $t \in V$

- 1 // Phase I
- 2  $\pi'(s, v)$  for all  $v \in V \leftarrow$  Monte Carlo estimates with  $\lceil 12 \ln(2n^3) / \varepsilon \rceil$  random walks
- 3  $C \leftarrow \{t \in V : \pi'(s, t) > \frac{1}{2}\varepsilon\}$
- 4 // Phase II
- 5  $n_r \leftarrow \lceil n/\varepsilon \rceil, n_t \leftarrow \lceil 18 \ln(2n^2) \rceil$
- 6  $r_{\max}(t) \leftarrow \frac{\varepsilon^2 n_r}{6\pi'(s, t)}$  for all  $t \in C$
- 7 **while** True **do**
- 8     // the process in this loop is called an iteration
- 9     // in this iteration, try running Backward Push with  $\frac{1}{2}r_{\max}(t)$ 's
- 10    **for** each  $t \in C$  **do**
- 11     run **BackwardPush** ( $G, \alpha, t, \frac{1}{2}r_{\max}(t)$ ), but once the total cost of Backward Push in this iteration exceeds the expected cost of simulating  $n_t \cdot \frac{1}{2}n_r$  random walks, terminate and break the outer loop (i.e., jump to line 14)
- 12      $n_r \leftarrow \lfloor \frac{1}{2}n_r \rfloor$
- 13      $r_{\max}(t) \leftarrow \frac{1}{2}r_{\max}(t)$  for all  $t \in C$
- 14  $q(v, t), r(v, t)$  for  $v \in V \leftarrow$  **BackwardPush**( $G, \alpha, t, r_{\max}(t)$ )
- 15 // Phase III
- 16 **for**  $i$  from 1 to  $n_t$  **do**
- 17      $\pi''(s, v)$  for all  $v \in V \leftarrow$  Monte Carlo estimates with  $n_r$  random walks
- 18      $\hat{\pi}_i(s, t) \leftarrow q(s, t) + \sum_{v \in V} \pi''(s, v)r(v, t)$  for all  $t \in C$
- 19  $\hat{\pi}(s, t) \leftarrow \text{median}_{i=1}^{n_t} \{\hat{\pi}_i(s, t)\}$  for all  $t \in C$
- 20 **return**  $\hat{\pi}(s, t)$  for all  $t \in V$

---

Concretely, in Phase I, the algorithm runs Monte Carlo (as introduced in Subsection 1.2) with  $\lceil 12 \ln(2n^3) / \varepsilon \rceil$  random walks (line 2), obtain estimates  $\pi'(s, v)$ , and sets the candidate set  $C$  to be  $\{t \in V : \pi'(s, t) > \frac{1}{2}\varepsilon\}$  (line 3). Next, Phase II implements Adaptive Backward Push. It first initializes  $n_r$ , the number of random walk samplings in Phase III, to be  $\lceil n/\varepsilon \rceil$ , and sets  $n_t$ , the number of trials for the median trick, to be  $\lceil 18 \ln(2n^2) \rceil$  (line 5). Also,  $r_{\max}(t)$  is initialized to be  $\frac{\varepsilon^2 n_r}{6\pi'(s, t)}$  for each candidate node  $t \in C$  (line 6). In the subsequent loop (line 7 to line 13), the algorithm repeatedly tries to run Backward Push (Algorithm 1) for each candidate node  $t \in C$  with parameter  $\frac{1}{2}r_{\max}(t)$ , and iteratively halves  $n_r$  as well as  $r_{\max}$  until the cost of Backward Push exceeds the expected cost of simulating random walks in Phase III. In that case, the algorithm immediately terminates Backward Push and jumps to line 14 (without halving  $n_r$  and  $r_{\max}(t)$ ), where Backward Push is invoked again with the current  $r_{\max}(t)$  to obtain the reserves and residues. Finally, in Phase III, the algorithm runs Monte Carlo with  $n_r$  random walks (line 17) and computes estimates  $\hat{\pi}_i(s, t)$  according to Equation 3 (line 18). This process is repeated  $n_t$  times (line 16), and the final approximations  $\hat{\pi}(s, t)$  are computed as  $\text{median}_{i=1}^{n_t} \{\hat{\pi}_i(s, t)\}$  (line 19).

Note that in line 18, for each  $t \in C$ , we only need to iterate through nodes  $v$  with a nonzero residue  $r(v, t)$  to compute the summation. Thus, we can implement line 18 in asymptotically the same time as running Backward Push in Phase II.

In the following section, we demonstrate the correctness and efficiency of Algorithm 2.

## 5 Analyses for the SSPPR-A Query

We give correctness and complexity analyses of our Algorithm 2 for the SSPPR-A query. The analyses for the SSPPR-D query are given in the full version of this paper [50].

First, we give error bounds for the Monte Carlo estimates in Phase I. These are typical results for the Monte Carlo method, and we give a proof in the full version of this paper [50] for completeness.

► **Lemma 4.** *Let  $\pi'(s, v)$  denote the estimate for  $\pi(s, v)$  obtained in Phase I of Algorithm 2. With probability at least  $1 - 1/n^2$ , we have  $\frac{1}{2}\pi(s, v) \leq \pi'(s, v) \leq \frac{3}{2}\pi(s, v)$  for all  $v \in V$  with  $\pi(s, v) \geq \frac{1}{4}\varepsilon$ , and  $\pi'(s, v) \leq \pi(s, v) + \frac{1}{4}\varepsilon$  for all  $v \in V$  with  $\pi(s, v) < \frac{1}{4}\varepsilon$ .*

We say that Phase I *succeeds* if the properties in Lemma 4 are satisfied. Our following discussions are implicitly conditioned on the success of Phase I, and we shall not specify this explicitly for ease of presentation. We only take this condition into account when considering the overall success probability of the algorithm. Also, we regard  $\pi'(s, v)$  as fixed values when analyzing the remaining phases. Next, we show that Phase I prunes non-candidate nodes properly and guarantees constant relative error bounds for candidate nodes.

► **Lemma 5.** *All non-candidate nodes  $t' \notin C$  satisfy  $\pi(s, t') \leq \varepsilon$ , and all candidate nodes  $t \in C$  satisfy  $\frac{1}{2}\pi(s, t) \leq \pi'(s, t) \leq \frac{3}{2}\pi(s, t)$ .*

The proof of Lemma 5 can be found in the full version of this paper [50]. Based on Lemma 5, we prove the correctness and complexity bounds of Algorithm 2 separately.

### Correctness Analysis

Primarily, the following lemma justifies the unbiasedness of the estimators  $\hat{\pi}_i(s, t)$ :

► **Lemma 6.** *For all the candidate nodes  $t \in C$ ,  $\hat{\pi}_i(s, t)$  are unbiased estimators for  $\pi(s, t)$ , i.e.,  $\mathbb{E}[\hat{\pi}_i(s, t)] = \pi(s, t)$ .*

The proof of Lemma 6 can be found in the full version of this paper [50]. Next, we prove the following key lemma, which bounds the variances of the estimators  $\hat{\pi}_i(s, t)$  in terms of  $n_r$ ,  $r_{\max}(t)$ , and  $\pi(s, t)$ :

► **Lemma 7.** *The variances  $\text{Var}[\hat{\pi}_i(s, t)]$  are bounded above by  $1/n_r \cdot r_{\max}(t)\pi(s, t)$ , for all  $t \in C$ . Here,  $n_r$  is the number of random walks in Phase III. This leads to  $\text{Var}[\hat{\pi}_i(s, t)] \leq \frac{1}{3}\varepsilon^2$ .*

**Proof.** We first calculate

$$\text{Var}[\hat{\pi}_i(s, t)] = \text{Var}\left[q(s, t) + \sum_{v \in V} \pi''(s, v)r(v, t)\right] = \text{Var}\left[\sum_{v \in V} \pi''(s, v)r(v, t)\right].$$

To bound this variance, we use the fact that the Monte Carlo estimators  $\pi'(s, v)$ 's are *negatively correlated*, so the variance of their weighted sum is bounded by the sum of their weighted variances:

$$\text{Var}[\hat{\pi}_i(s, t)] \leq \sum_{v \in V} \text{Var}[\pi''(s, v)r(v, t)] = \sum_{v \in V} (r(v, t))^2 \text{Var}[\pi''(s, v)].$$



Next, we plug in  $\text{Var} [\pi''(s, v)] = \pi(s, v)(1 - \pi(s, v))/n_r$ , the variances of binomial random variables divided by  $n_r$ , to obtain

$$\begin{aligned} \text{Var} [\hat{\pi}_i(s, t)] &\leq \sum_{v \in V} (r(v, t))^2 \cdot \frac{\pi(s, v)(1 - \pi(s, v))}{n_r} \\ &= \frac{1}{n_r} \sum_{v \in V} r(v, t)(\pi(s, v)r(v, t))(1 - \pi(s, v)) \leq \frac{1}{n_r} \sum_{v \in V} r(v, t)(\pi(s, v)r(v, t)). \end{aligned}$$

Using the properties of Backward Push (see Subsection 3.2) that  $r(v, t) \leq r_{\max}(t)$  for all  $v \in V$  and  $\sum_{v \in V} \pi(s, v)r(v, t) = \pi(s, t) - q(s, t) \leq \pi(s, t)$ , we have

$$\text{Var} [\hat{\pi}_i(s, t)] \leq \frac{1}{n_r} \cdot r_{\max}(t) \sum_{v \in V} \pi(s, v)r(v, t) \leq \frac{1}{n_r} \cdot r_{\max}(t)\pi(s, t).$$

This proves the first part of the lemma. Finally, by the setting of  $r_{\max}(t) = \frac{\varepsilon^2 n_r}{6\pi'(s, t)}$  and the result in Lemma 5 that  $\pi'(s, t) \geq \frac{1}{2}\pi(s, t)$  for candidate nodes  $t \in C$ , we obtain

$$\text{Var} [\hat{\pi}_i(s, t)] \leq \frac{1}{n_r} \cdot \frac{\varepsilon^2 n_r}{6\pi'(s, t)} \cdot \pi(s, t) \leq \frac{1}{n_r} \cdot \frac{\varepsilon^2 n_r}{3\pi(s, t)} \cdot \pi(s, t) = \frac{1}{3}\varepsilon^2.$$

We conclude that  $\text{Var} [\hat{\pi}_i(s, t)] \leq \frac{1}{3}\varepsilon^2$ , as claimed.  $\blacktriangleleft$

Now, we prove the following theorem, which verifies the correctness of Algorithm 2:

**► Theorem 8.** *Algorithm 2 answers the SSPPR-A query (defined in Definition 1) correctly with probability at least  $1 - 1/n$ .*

**Proof.** First, for non-candidate nodes  $t' \notin C$ , Lemma 5 guarantees that  $\pi(s, t') \leq \varepsilon$ , so it is acceptable that Algorithm 2 returns  $\hat{\pi}(s, t') = 0$  as their PPR estimates. For candidate nodes  $t \in C$ , Lemma 6 together with Chebyshev's inequality guarantees that

$$\Pr \left[ |\hat{\pi}_i(s, t) - \pi(s, t)| \geq \varepsilon \right] \leq \frac{\text{Var} [\hat{\pi}_i(s, t)]}{\varepsilon^2} \leq \frac{1}{3}.$$

Recall that Algorithm 2 sets the final estimates  $\hat{\pi}(s, t)$  to be  $\text{median}_{i=1}^{n_t} \{\hat{\pi}_i(s, t)\}$ , where  $n_t = \lceil 18 \ln(2n^2) \rceil$ , and that  $\hat{\pi}_i(s, t)$  for  $1 \leq i \leq n_t$  are obtained from independent trials of  $n_r$  random walk samplings. Thus, by applying the median trick (Theorem 15), we know that for any  $t \in C$ , the probability that  $|\hat{\pi}(s, t) - \pi(s, t)| \geq \varepsilon$  is at most  $1/(2n^2)$ . Since  $|C| \leq n$ , by applying union bound for all  $t \in C$ , we prove that with probability at least  $1 - 1/(2n)$ ,  $|\hat{\pi}(s, t) - \pi(s, t)| \leq \varepsilon$  holds for all  $t \in C$ , matching the error bound required in Definition 1. Lastly, recall that this probability is conditioned on the success of Phase I, whose probability is at least  $1 - 1/n^2$  by Lemma 4. Thus, we conclude that the overall success probability is at least  $(1 - 1/n^2)(1 - 1/(2n)) > 1 - 1/n$ .  $\blacktriangleleft$

### Complexity Analysis

First, we formalize the claim given in Subsection 4.2 regarding the balance between Phases II and III, as follows. We prove the claim in the full version of this paper [50].

**▷ Claim 9.** Algorithm 2 achieves the asymptotic complexity as if the optimal  $n_r$  (in terms of balancing the complexities of Phases II and III) is previously known and Adaptive Backward Push is only performed once with this  $n_r$ .

## 9:14 Approximating Single-Source PPR with Absolute Error Guarantees

Based on Claim 9, we can derive the complexity of Algorithm 2 on general directed graphs:

► **Theorem 10.** *The expected time complexity of Algorithm 2 on directed graphs is*

$$\tilde{O}\left(\frac{1}{\varepsilon}\sqrt{\sum_{t \in V} \pi(s, t) \sum_{v \in V} \pi(v, t) d_{\text{in}}(v)}\right).$$

Furthermore, this complexity is upper bounded by  $\tilde{O}(\sqrt{m}/\varepsilon)$ .

**Proof.** First, the expected complexity of Phase I is  $\tilde{O}(1/\varepsilon)$ , which is negligible in the overall complexity. In Phase II, Backward Push is invoked with parameter  $r_{\max}(t) = \frac{\varepsilon^2 n_r}{6\pi'(s, t)}$  for each  $t \in C$ , where  $\pi'(s, t)$  satisfies  $\pi'(s, t) \leq \frac{3}{2}\pi(s, t)$  by Lemma 5. Therefore, using the complexity of Backward Push (Equation 2), the complexity of Phase II is bounded by

$$\begin{aligned} & O\left(\sum_{t \in C} \frac{\sum_{v \in V} \pi(v, t) d_{\text{in}}(v)}{r_{\max}(t)}\right) = O\left(\sum_{t \in C} \sum_{v \in V} \frac{\pi'(s, t) \pi(v, t) d_{\text{in}}(v)}{\varepsilon^2 n_r}\right) \\ & \leq O\left(\sum_{t \in C} \sum_{v \in V} \frac{\pi(s, t) \pi(v, t) d_{\text{in}}(v)}{\varepsilon^2 n_r}\right) \leq O\left(\sum_{t \in V} \sum_{v \in V} \frac{\pi(s, t) \pi(v, t) d_{\text{in}}(v)}{\varepsilon^2 n_r}\right) \\ & = O\left(\frac{1}{\varepsilon^2 n_r} \sum_{t \in V} \pi(s, t) \sum_{v \in V} \pi(v, t) d_{\text{in}}(v)\right). \end{aligned} \quad (4)$$

On the other hand, Phase III performs  $n_t \cdot n_r = \lceil 18 \ln(2n^2) \rceil \cdot n_r = \tilde{O}(n_r)$  random walks, so the total complexity of Phases II and III is  $\tilde{O}(1/(\varepsilon^2 n_r) \cdot \sum_{t \in V} \pi(s, t) \sum_{v \in V} \pi(v, t) d_{\text{in}}(v) + n_r)$ . By the AM–GM inequality, the optimal setting of  $n_r$  minimizes this bound to be

$$\tilde{O}\left(\sqrt{n_r \cdot \frac{1}{\varepsilon^2 n_r} \sum_{t \in V} \pi(s, t) \sum_{v \in V} \pi(v, t) d_{\text{in}}(v)}\right) = \tilde{O}\left(\frac{1}{\varepsilon} \sqrt{\sum_{t \in V} \pi(s, t) \sum_{v \in V} \pi(v, t) d_{\text{in}}(v)}\right).$$

By Claim 9, Algorithm 2 achieves this complexity.

In order for a simple upper bound, we use  $\pi(v, t) \leq 1$  for all  $v, t \in V$  to obtain

$$\begin{aligned} & \tilde{O}\left(\frac{1}{\varepsilon} \sqrt{\sum_{t \in V} \pi(s, t) \sum_{v \in V} \pi(v, t) d_{\text{in}}(v)}\right) \leq \tilde{O}\left(\frac{1}{\varepsilon} \sqrt{\sum_{t \in V} \pi(s, t) \sum_{v \in V} d_{\text{in}}(v)}\right) \\ & = \tilde{O}\left(\frac{1}{\varepsilon} \sqrt{\sum_{t \in V} \pi(s, t) \cdot m}\right) = \tilde{O}\left(\frac{\sqrt{m}}{\varepsilon}\right), \end{aligned}$$

as claimed.

A subtle technicality here is that these complexity bounds are conditioned on the success of Phase I, and the expected complexity of Phase II may be unbounded if Phase I fails. To resolve this technical issue, we can switch to using naïve Power Method [14] once the actual cost of the algorithm reaches  $\Theta(n^2)$ . In this way, even if Phase I fails, the algorithm will solve the query in  $\tilde{O}(n^2)$  time. As the failure probability of Phase I is at most  $1/n^2$  (Lemma 4), this merely adds a term of  $1/n^2 \cdot \tilde{O}(n^2) = \tilde{O}(1)$  to the overall expected complexity, and thus does not affect the resultant bounds. We will omit this technicality in later proofs. ◀

Next, we analyze the complexity of Algorithm 2 on general undirected graphs. To this end, the following previously known symmetry theorem will be helpful.

► **Theorem 11** (Symmetry of PPR on Undirected Graphs [6, Lemma 1]). *For all nodes  $u, v \in V$ , we have  $\pi(u, v)d(u) = \pi(v, u)d(v)$ .*

Now, we can derive a better complexity bound of Algorithm 2 on undirected graphs.

► **Theorem 12.** *The expected time complexity of Algorithm 2 on undirected graphs is*

$$\tilde{O}\left(\frac{1}{\varepsilon}\sqrt{\sum_{t \in V} \pi(s, t)d(t)}\right).$$

Furthermore, this complexity is upper bounded by  $\tilde{O}(\sqrt{d_{\max}}/\varepsilon)$ .

**Proof.** Using Theorem 11, we can simplify the first complexity in Theorem 10 as follows:

$$\begin{aligned} & \tilde{O}\left(\frac{1}{\varepsilon}\sqrt{\sum_{t \in V} \pi(s, t) \sum_{v \in V} \pi(v, t)d(v)}\right) = \tilde{O}\left(\frac{1}{\varepsilon}\sqrt{\sum_{t \in V} \pi(s, t) \sum_{v \in V} \pi(t, v)d(t)}\right) \\ & = \tilde{O}\left(\frac{1}{\varepsilon}\sqrt{\sum_{t \in V} \pi(s, t)d(t) \sum_{v \in V} \pi(t, v)}\right) = \tilde{O}\left(\frac{1}{\varepsilon}\sqrt{\sum_{t \in V} \pi(s, t)d(t)}\right). \end{aligned}$$

To prove the upper bound of  $\tilde{O}(\sqrt{d_{\max}}/\varepsilon)$ , we use  $d(t) \leq d_{\max}$  for all  $t \in V$  to obtain

$$\tilde{O}\left(\frac{1}{\varepsilon}\sqrt{\sum_{t \in V} \pi(s, t)d(t)}\right) \leq \tilde{O}\left(\frac{\sqrt{d_{\max}}}{\varepsilon} \cdot \sqrt{\sum_{t \in V} \pi(s, t)}\right) = \tilde{O}\left(\frac{\sqrt{d_{\max}}}{\varepsilon}\right). \quad \blacktriangleleft$$

We note that the bounds  $\tilde{O}(\sqrt{m}/\varepsilon)$  and  $\tilde{O}(\sqrt{d_{\max}}/\varepsilon)$  above are for worst-case graphs, and for power-law graphs (Assumption 3), we can derive better bounds. In the full version of this paper [50], we prove the following lemma, which bounds  $\sum_{t \in V} (\pi(v, t))^2$  for any  $v \in V$ :

► **Lemma 13.** *On a power-law graph,  $\sum_{t \in V} (\pi(v, t))^2 = O(n^{2\gamma-2})$  holds for any  $v \in V$ .*

Now, we can bound the complexity on power-law graphs as follows:

► **Theorem 14.** *The expected complexity of Algorithm 2 on power-law graphs is  $\tilde{O}(n^{\gamma-1/2}/\varepsilon)$ .*

**Proof.** We prove the theorem using *Cauchy–Schwarz inequality* and power-law assumptions to simplify the first complexity given in Theorem 10. First, reordering the summations yields

$$\tilde{O}\left(\frac{1}{\varepsilon}\sqrt{\sum_{t \in V} \pi(s, t) \sum_{v \in V} \pi(v, t)d_{\text{in}}(v)}\right) = \tilde{O}\left(\frac{1}{\varepsilon}\sqrt{\sum_{v \in V} d_{\text{in}}(v) \sum_{t \in V} \pi(s, t)\pi(v, t)}\right),$$

where

$$\sum_{t \in V} \pi(s, t)\pi(v, t) \leq \sqrt{\left(\sum_{t \in V} (\pi(s, t))^2\right) \left(\sum_{t \in V} (\pi(v, t))^2\right)}$$

by Cauchy–Schwarz inequality. By Lemma 13, both  $\sum_{t \in V} (\pi(s, t))^2$  and  $\sum_{t \in V} (\pi(v, t))^2$  are bounded by  $O(n^{2\gamma-2})$ . Consequently,  $\sum_{t \in V} \pi(s, t)\pi(v, t) \leq O(n^{2\gamma-2})$ , and the expression in question can be bounded by

$$\tilde{O}\left(\frac{1}{\varepsilon}\sqrt{\sum_{v \in V} d_{\text{in}}(v) \cdot n^{2\gamma-2}}\right) = \tilde{O}\left(\frac{1}{\varepsilon}\sqrt{m \cdot n^{2\gamma-2}}\right) = \tilde{O}\left(\frac{n^{\gamma-1/2}}{\varepsilon}\right),$$

where we used the fact that  $m = \tilde{O}(n)$  on power-law graphs. ◀

## References

- 1 Reid Andersen, Christian Borgs, Jennifer T. Chayes, John E. Hopcroft, Vahab S. Mirrokni, and Shang-Hua Teng. Local computation of pagerank contributions. In *Proc. 5th Int. Workshop Algorithms Models Web Graph*, volume 4863, pages 150–165, 2007. doi:10.1007/978-3-540-77004-6\_12.
- 2 Reid Andersen, Christian Borgs, Jennifer T. Chayes, John E. Hopcroft, Vahab S. Mirrokni, and Shang-Hua Teng. Local computation of pagerank contributions. *Internet Math.*, 5(1):23–45, 2008. doi:10.1080/15427951.2008.10129302.
- 3 Reid Andersen and Fan R. K. Chung. Detecting sharp drops in pagerank and a simplified local partitioning algorithm. In *Proc. 4th Int. Conf. Theory Appl. Models Comput.*, volume 4484, pages 1–12, 2007. doi:10.1007/978-3-540-72504-6\_1.
- 4 Reid Andersen, Fan R. K. Chung, and Kevin J. Lang. Local graph partitioning using pagerank vectors. In *Proc. 47th Annu. IEEE Symp. Found. Comput. Sci.*, pages 475–486, 2006. doi:10.1109/FOCS.2006.44.
- 5 Reid Andersen, Fan R. K. Chung, and Kevin J. Lang. Using pagerank to locally partition a graph. *Internet Math.*, 4(1):35–64, 2007. doi:10.1080/15427951.2007.10129139.
- 6 Konstantin Avrachenkov, Paulo Gonçalves, and Marina Sokol. On the choice of kernel and labelled data in semi-supervised learning methods. In *Proc. 10th Int. Workshop Algorithms Models Web Graph*, volume 8305, pages 56–67, 2013. doi:10.1007/978-3-319-03536-9\_5.
- 7 Konstantin Avrachenkov, Nelly Litvak, Danil Nemirovsky, Elena Smirnova, and Marina Sokol. Quick detection of top-k personalized pagerank lists. In *Proc. 8th Int. Workshop Algorithms Models Web Graph*, volume 6732, pages 50–61, 2011. doi:10.1007/978-3-642-21286-4\_5.
- 8 Bahman Bahmani, Kaushik Chakrabarti, and Dong Xin. Fast personalized pagerank on mapreduce. In *Proc. ACM SIGMOD Int. Conf. Manage. Data*, pages 973–984, 2011. doi:10.1145/1989323.1989425.
- 9 Bahman Bahmani, Abdur Chowdhury, and Ashish Goel. Fast incremental and personalized pagerank. *Proc. VLDB Endowment*, 4(3):173–184, 2010. doi:10.14778/1929861.1929864.
- 10 Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999. doi:10.1126/science.286.5439.509.
- 11 Pavel Berkhin. Bookmark-coloring algorithm for personalized pagerank computing. *Internet Math.*, 3(1):41–62, 2006. doi:10.1080/15427951.2006.10129116.
- 12 Aleksandar Bojchevski, Johannes Klicpera, Bryan Perozzi, Amol Kapoor, Martin Blais, Benedek Rózemberczki, Michal Lukasik, and Stephan Günnemann. Scaling graph neural networks with approximate pagerank. In *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, pages 2464–2473, 2020. doi:10.1145/3394486.3403296.
- 13 Béla Bollobás, Christian Borgs, Jennifer T. Chayes, and Oliver Riordan. Directed scale-free graphs. In *Proc. ACM-SIAM Symp. Discrete Algorithms*, pages 132–139, 2003. URL: <http://dl.acm.org/citation.cfm?id=644108.644133>.
- 14 Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Comput. Netw.*, 30(1-7):107–117, 1998. doi:10.1016/S0169-7552(98)00110-X.
- 15 Mustafa Coşkun, Ananth Grama, and Mehmet Koyutürk. Efficient processing of network proximity queries via chebyshev acceleration. In *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, pages 1515–1524, 2016. doi:10.1145/2939672.2939828.
- 16 Dániel Fogaras, Balázs Rácz, Károly Csalogány, and Tamás Sarlós. Towards scaling fully personalized pagerank: Algorithms, lower bounds, and experiments. *Internet Math.*, 2(3):333–358, 2005. doi:10.1080/15427951.2005.10129104.
- 17 Kimon Fountoulakis, Farbod Roosta-Khorasani, Julian Shun, Xiang Cheng, and Michael W. Mahoney. Variational perspective on local graph clustering. *Math. Program.*, 174(1-2):553–573, 2019. doi:10.1007/S10107-017-1214-8.
- 18 Yasuhiro Fujiwara, Makoto Nakatsuji, Makoto Onizuka, and Masaru Kitsuregawa. Fast and exact top-k search for random walk with restart. *Proc. VLDB Endowment*, 5(5):442–453, 2012. doi:10.14778/2140436.2140441.

- 19 Yasuhiro Fujiwara, Makoto Nakatsuji, Hiroaki Shiokawa, Takeshi Mishima, and Makoto Onizuka. Efficient ad-hoc search for personalized pagerank. In *Proc. ACM SIGMOD Int. Conf. Manage. Data*, pages 445–456, 2013. doi:10.1145/2463676.2463717.
- 20 Yasuhiro Fujiwara, Makoto Nakatsuji, Takeshi Yamamuro, Hiroaki Shiokawa, and Makoto Onizuka. Efficient personalized pagerank with accuracy assurance. In *Proc. 18th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, pages 15–23, 2012. doi:10.1145/2339530.2339538.
- 21 David F. Gleich. Pagerank beyond the web. *SIAM Rev.*, 57(3):321–363, 2015. doi:10.1137/140976649.
- 22 Tao Guo, Xin Cao, Gao Cong, Jiaheng Lu, and Xuemin Lin. Distributed algorithms on exact personalized pagerank. In *Proc. ACM SIGMOD Int. Conf. Manage. Data*, pages 479–494, 2017. doi:10.1145/3035918.3035920.
- 23 Wentian Guo, Yuchen Li, Mo Sha, and Kian-Lee Tan. Parallel personalized pagerank on dynamic graphs. *Proc. VLDB Endowment*, 11(1):93–106, 2017. doi:10.14778/3151113.3151121.
- 24 Guanhao Hou, Xingguang Chen, Sibao Wang, and Zhewei Wei. Massively parallel algorithms for personalized pagerank. *Proc. VLDB Endowment*, 14(9):1668–1680, 2021. doi:10.14778/3461535.3461554.
- 25 Mark Jerrum, Leslie G. Valiant, and Vijay V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theor. Comput. Sci.*, 43:169–188, 1986. doi:10.1016/0304-3975(86)90174-X.
- 26 Jinhong Jung, Namyoung Park, Lee Sael, and U Kang. Bepi: Fast and memory-efficient method for billion-scale random walk with restart. In *Proc. ACM SIGMOD Int. Conf. Manage. Data*, pages 789–804, 2017. doi:10.1145/3035918.3035950.
- 27 Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. In *Proc. 7th Int. Conf. Learn. Representations*, 2019. URL: <https://openreview.net/forum?id=H1gL-2A9Ym>.
- 28 Meihao Liao, Rong-Hua Li, Qiangqiang Dai, Hongyang Chen, Hongchao Qin, and Guoren Wang. Efficient personalized pagerank computation: The power of variance-reduced monte carlo approaches. *Proc. ACM Manage. Data*, 1(2):160:1–160:26, 2023. doi:10.1145/3589305.
- 29 Meihao Liao, Rong-Hua Li, Qiangqiang Dai, and Guoren Wang. Efficient personalized pagerank computation: A spanning forests sampling based approach. In *Proc. ACM SIGMOD Int. Conf. Manage. Data*, pages 2048–2061, 2022. doi:10.1145/3514221.3526140.
- 30 Dandan Lin, Raymond Chi-Wing Wong, Min Xie, and Victor Junqiu Wei. Index-free approach with theoretical guarantee for efficient random walk with restart query. In *Proc. 36th Int. Conf. Data Eng.*, pages 913–924, 2020. doi:10.1109/ICDE48307.2020.00084.
- 31 Wenqing Lin. Distributed algorithms for fully personalized pagerank on large graphs. In *Proc. Int. Conf. World Wide Web*, pages 1084–1094, 2019. doi:10.1145/3308558.3313555.
- 32 Peter Lofgren, Siddhartha Banerjee, and Ashish Goel. Personalized pagerank estimation and search: A bidirectional approach. In *Proc. 9th ACM Int. Conf. Web Search Data Mining*, pages 163–172, 2016. doi:10.1145/2835776.2835823.
- 33 Peter Lofgren, Siddhartha Banerjee, Ashish Goel, and Seshadhri Comandur. Fast-ppr: scaling personalized pagerank estimation for large graphs. In *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, pages 1436–1445, 2014. doi:10.1145/2623330.2623745.
- 34 Peter Lofgren and Ashish Goel. Personalized pagerank to a target node. *CoRR*, abs/1304.4658, 2013. doi:10.48550/arXiv.1304.4658.
- 35 Takanori Maehara, Takuya Akiba, Yoichi Iwata, and Ken-ichi Kawarabayashi. Computing personalized pagerank quickly by exploiting graph structures. *Proc. VLDB Endowment*, 7(12):1023–1034, 2014. doi:10.14778/2732977.2732978.
- 36 Naoto Ohsaka, Takanori Maehara, and Ken-ichi Kawarabayashi. Efficient pagerank tracking in evolving networks. In *Proc. 21st ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, pages 875–884, 2015. doi:10.1145/2783258.2783297.

- 37 Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric transitivity preserving graph embedding. In *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, pages 1105–1114, 2016. doi:10.1145/2939672.2939751.
- 38 Jieming Shi, Renchi Yang, Tianyuan Jin, Xiaokui Xiao, and Yin Yang. Realtime top-k personalized pagerank over large graphs on gpus. *Proc. VLDB Endowment*, 13(1):15–28, 2019. doi:10.14778/3357377.3357379.
- 39 Kijung Shin, Jinhong Jung, Lee Sael, and U Kang. Bear: Block elimination approach for random walk with restart on large graphs. In *Proc. ACM SIGMOD Int. Conf. Manage. Data*, pages 1571–1585, 2015. doi:10.1145/2723372.2723716.
- 40 Anton Tsitsulin, Davide Mottin, Panagiotis Karras, and Emmanuel Müller. Verse: Versatile graph embeddings from similarity measures. In *Proc. Int. Conf. World Wide Web*, pages 539–548, 2018. doi:10.1145/3178876.3186120.
- 41 Alastair J Walker. New fast method for generating discrete random numbers with arbitrary frequency distributions. *Electronics Letters*, 8(10):127–128, 1974. doi:10.1049/e1:19740097.
- 42 Hanzhi Wang, Mingguo He, Zhewei Wei, Sibowang, Ye Yuan, Xiaoyong Du, and Ji-Rong Wen. Approximate graph propagation. In *Proc. 27th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, pages 1686–1696, 2021. doi:10.1145/3447548.3467243.
- 43 Hanzhi Wang, Zhewei Wei, Junhao Gan, Sibowang, and Zengfeng Huang. Personalized pagerank to a target node, revisited. In *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, pages 657–667, 2020. doi:10.1145/3394486.3403108.
- 44 Runhui Wang, Sibowang, and Xiaofang Zhou. Parallelizing approximate single-source personalized pagerank queries on shared memory. *VLDB J.*, 28(6):923–940, 2019. doi:10.1007/S00778-019-00576-7.
- 45 Sibowang, Youze Tang, Xiaokui Xiao, Yin Yang, and Zengxiang Li. Hubppr: Effective indexing for approximate personalized pagerank. *Proc. VLDB Endowment*, 10(3):205–216, 2016. doi:10.14778/3021924.3021936.
- 46 Sibowang, Renchi Yang, Runhui Wang, Xiaokui Xiao, Zhewei Wei, Wenqing Lin, Yin Yang, and Nan Tang. Efficient algorithms for approximate single-source personalized pagerank queries. *ACM Trans. Database Syst.*, 44(4):18:1–18:37, 2019. doi:10.1145/3360902.
- 47 Sibowang, Renchi Yang, Xiaokui Xiao, Zhewei Wei, and Yin Yang. Fora: Simple and effective approximate single-source personalized pagerank. In *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, pages 505–514, 2017. doi:10.1145/3097983.3098072.
- 48 Zhewei Wei, Xiaodong He, Xiaokui Xiao, Sibowang, Yu Liu, Xiaoyong Du, and Ji-Rong Wen. Prsim: Sublinear time simrank computation on large power-law graphs. In *Proc. ACM SIGMOD Int. Conf. Manage. Data*, pages 1042–1059, 2019. doi:10.1145/3299869.3319873.
- 49 Zhewei Wei, Xiaodong He, Xiaokui Xiao, Sibowang, Shuo Shang, and Ji-Rong Wen. Topppr: Top-k personalized pagerank queries with precision guarantees on large graphs. In *Proc. ACM SIGMOD Int. Conf. Manage. Data*, pages 441–456, 2018. doi:10.1145/3183713.3196920.
- 50 Zhewei Wei, Ji-Rong Wen, and Mingji Yang. Approximating single-source personalized pagerank with absolute error guarantees. *CoRR*, abs/2401.01019, 2024. doi:10.48550/arXiv.2401.01019.
- 51 Hao Wu, Junhao Gan, Zhewei Wei, and Rui Zhang. Unifying the global and local approaches: An efficient power iteration with forward push. In *Proc. ACM SIGMOD Int. Conf. Manage. Data*, pages 1996–2008, 2021. doi:10.1145/3448016.3457298.
- 52 Yubao Wu, Ruoming Jin, and Xiang Zhang. Fast and unified local search for random walk based k-nearest-neighbor query in large graphs. In *Proc. ACM SIGMOD Int. Conf. Manage. Data*, pages 1139–1150, 2014. doi:10.1145/2588555.2610500.
- 53 Hao Yin, Austin R. Benson, Jure Leskovec, and David F. Gleich. Local higher-order graph clustering. In *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, pages 555–564, 2017. doi:10.1145/3097983.3098069.



- 54 Yuan Yin and Zhewei Wei. Scalable graph embeddings via sparse transpose proximities. In *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, pages 1429–1437, 2019. doi:10.1145/3292500.3330860.
- 55 Minji Yoon, Woojeong Jin, and U Kang. Fast and accurate random walk with restart on dynamic graphs with guarantees. In *Proc. Int. Conf. World Wide Web*, pages 409–418, 2018. doi:10.1145/3178876.3186107.
- 56 Minji Yoon, Jinhong Jung, and U Kang. TPA: Fast, scalable, and accurate method for approximate random walk with restart on billion scale graphs. In *Proc. 34th Int. Conf. Data Eng.*, pages 1132–1143, 2018. doi:10.1109/ICDE.2018.00105.
- 57 Weiren Yu and Xuemin Lin. IRWR: Incremental random walk with restart. In *Proc. 36th ACM SIGIR Int. Conf. Res. Develop. Inf. Retrieval*, pages 1017–1020, 2013. doi:10.1145/2484028.2484114.
- 58 Weiren Yu and Julie A. McCann. Random walk with restart over dynamic graphs. In *Proc. 16th Int. Conf. Data Mining*, pages 589–598, 2016. doi:10.1109/ICDM.2016.0070.
- 59 Fanwei Zhu, Yuan Fang, Kevin Chen-Chuan Chang, and Jing Ying. Incremental and accuracy-aware personalized pagerank through scheduled approximation. *Proc. VLDB Endowment*, 6(6):481–492, 2013. doi:10.14778/2536336.2536348.

## A

 Median Trick

► **Theorem 15** (Median Trick [25]). Let  $X_1, X_2, \dots, X_{n_t}$  be  $n_t$  i.i.d. random variables such that  $\Pr[|X_i - \mu| \geq \lambda] \leq \frac{1}{3}$  for any  $1 \leq i \leq n_t$ , where  $\mu = \mathbb{E}[X_i]$ , and let  $X = \text{median}_{1 \leq i \leq n_t} \{X_i\}$ . For a given probability  $p_f$ , if  $n_t \geq 18 \ln(1/p_f) = \Theta(\log(1/p_f))$ , then  $\Pr[|X - \mu| \geq \lambda] \leq p_f$ . Here,  $\text{median}_{1 \leq i \leq n_t} \{X_i\}$  is defined as the  $\lceil \frac{n_t}{2} \rceil$ -th smallest element in  $X_1, X_2, \dots, X_{n_t}$ .





# Right-Adjoint for Datalog Programs

Balder ten Cate  

Institute for Logic, Language, and Computation, University of Amsterdam, The Netherlands

Víctor Dalmau  

Department of Information and Communication Technologies, Universitat Pompeu Fabra, Barcelona, Spain

Jakub Opršal  

School of Computer Science, University of Birmingham, UK

---

## Abstract

A Datalog program can be viewed as a syntactic specification of a mapping from database instances over some schema to database instances over another schema. We establish a large class of Datalog programs for which this mapping admits a (generalized) right-adjoint. We employ these results to obtain new insights into the existence of, and methods for constructing, homomorphism dualities within restricted classes of instances. From this, we derive new results regarding the existence of uniquely characterizing data examples for database queries in the presence of integrity constraints.

**2012 ACM Subject Classification** Theory of computation → Logic and databases

**Keywords and phrases** Datalog, Adjoint, Homomorphism Dualities, Database Constraints, Conjunctive Queries, Data Examples

**Digital Object Identifier** 10.4230/LIPIcs.ICDT.2024.10

**Funding** *Balder ten Cate*: Supported by the European Union’s Horizon 2020 research and innovation programme (MSCA-101031081).

*Víctor Dalmau*: Supported by the MiCin under grants PID2019-109137GB-C22 and PID2022-138506NB-C22, and the Maria de Maeztu program (CEX2021-001195-M).

*Jakub Opršal*: Supported by the European Union’s Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie Grant Agreement No 101034413.

**Acknowledgements** Thanks to Pierre Bourrhis for detailed feedback and catching mistakes.

## 1 Introduction

Datalog is a rule-based language for specifying mappings from database instances over an input schema  $\mathbf{S}_{in}$ , to database instances over an output schema  $\mathbf{S}_{out}$ .

► **Example 1.1.** Consider the Datalog program defined by the following rules:

$$\begin{aligned} \text{Path}(x, y) &:- \text{Edge}(x, y). \\ \text{Path}(x, y) &:- \text{Edge}(x, z), \text{Path}(z, y). \\ \text{Ans}(x, y) &:- \text{Path}(x, y). \end{aligned}$$

This Datalog program takes as input an instance over an input schema  $\{\text{Edge}\}$ , and produces as output an instance over the schema  $\{\text{Ans}\}$ , where  $\text{Ans}$  is the transitive closure of  $\text{Edge}$ .

We study the existence of right-adjoints and generalized right-adjoints for Datalog programs, where a *right-adjoint* for a Datalog program  $P$  is a function  $\Omega$  from  $\mathbf{S}_{out}$ -instances to  $\mathbf{S}_{in}$ -instances, such that for all  $\mathbf{S}_{in}$ -instances  $I$  and  $\mathbf{S}_{out}$ -instances  $J$ ,  $P(I) \rightarrow J$  iff  $I \rightarrow \Omega(J)$ , where “ $\rightarrow$ ” denotes the existence of a homomorphism. *Generalized* right-adjoints are defined similarly, loosely speaking, except that we allow  $\Omega$  to map each  $\mathbf{S}_{out}$ -instance to a *finite set* of  $\mathbf{S}_{in}$ -instances, such that,  $P(I) \rightarrow J$  iff  $I \rightarrow J'$  for some  $J' \in \Omega(J)$ . We



© Balder ten Cate, Víctor Dalmau, and Jakub Opršal;  
licensed under Creative Commons License CC-BY 4.0

27th International Conference on Database Theory (ICDT 2024).

Editors: Graham Cormode and Michael Shekelyan; Article No. 10; pp. 10:1–10:20

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

## 10:2 Right-Adjoint for Datalog Programs

identify large classes of Datalog programs for which a right-adjoint, respectively, a generalized right-adjoint, exists. For instance, it will follow from our results that the Datalog program from Example 1.1 has a right-adjoint.

Our motivation for studying (generalized) right-adjoints for Datalog programs comes from the fact that they provide us with a means of constructing homomorphism dualities. A homomorphism duality is a pair  $(F, D)$  where  $F$  and  $D$  are sets of instances, such that an arbitrary instance  $A$  admits a homomorphism from an instance in  $F$  if and only if  $A$  does not admit a homomorphism to any instance in  $D$ . In other words, homomorphism dualities equate the existence of a homomorphism of one kind to the non-existence of a homomorphism of another kind. Homomorphism dualities have been studied extensively in the literature on constraint satisfaction problems, and have also found several applications in database theory (e.g., for schema mapping design [2], ontology-mediated data access [6], and query inference from data examples [9, 10]). In particular, in [2, 9], homomorphism dualities were used as a tool for studying the unique characterizability, and exact learnability, of schema mappings and of conjunctive queries. Using the same approach, we can use our results on right-adjoints to derive new results on unique characterizations for (unions of) conjunctive queries in the presence of integrity constraints. In the process, we also obtain a new technique for constructing homomorphism dualities within restricted classes of structures, e.g., transitive digraphs.

► **Contribution 1** (Section 3). We introduce a new fragment of Datalog called *TAM Datalog* (*Tree-Shaped Almost-Monadic Datalog*). We characterize TAM Datalog as a fragment of Monadic Second-Order Logic, and we prove that TAM Datalog is closed under composition.

► **Contribution 2** (Section 4). We show that every connected TAM Datalog program has a right-adjoint, and that every TAM Datalog program has a generalized right-adjoint. We show by means of counterexamples that each of the syntactic conditions imposed by TAM Datalog is necessary for the existence of generalized right-adjoints.

► **Contribution 3** (Section 5). We investigate the relationship between generalized right-adjoints and homomorphism dualities. Generalized right-adjoints can be used for constructing homomorphism dualities. We show that all tree dualities can be accounted for in this way.

► **Contribution 4** (Section 6). Following the approach in [2, 9], we derive new results on unique characterizations for (unions of) conjunctive queries in the presence of integrity constraints. In the process, we obtain a new technique for constructing homomorphism dualities within restricted classes of structures, e.g., transitive digraphs.

Some proofs are omitted and can be found in an appendix of the full version of this paper.

**Related Work.** Foniok and Tardif [17] studied the existence of right adjoints to Pultr functors which are themselves right adjoints [22] in the special case of digraphs. Translating into our terms a Pultr functor is an interpretation (of digraphs in digraphs)  $(\phi_V, \phi_E)$  where  $\phi_V$  and  $\phi_E$  are conjunctive queries (with  $k$  and  $2k$  free variables, respectively, for some  $k \geq 1$ ) defining the output node-set and edge-set respectively. For the special case where  $\phi_V$  just returns the input node-set, it was shown in [17] that the functor defined by  $(\phi_V, \phi_E)$  has a right adjoint if  $\phi_E$  is connected and acyclic. The setup and characterization were generalized in [13] to arbitrary relational structures. We extensively build on the framework and concepts in [13], but we permit the interpretation to be specified by an arbitrary Datalog program, so that our setup is able to encompass common types of database dependencies.

To our knowledge, this is the first time that adjoints for functors defined by Datalog programs have been studied. Also, it is the first application of functors with right adjoints in the context of unique characterization of database queries. In a different setting, namely approximate graph coloring, the “arc graph” functor was used in [21] where it is additionally argued that functors with a right adjoint, more generally, can play a role in the design and analysis of reductions between (promise) constraint satisfaction problems. The use of Datalog programs for reductions between such problems, although without reference to adjoints, is discussed in [14].

## 2 Preliminaries

**Schemas, Instances, Homomorphisms.** A *schema*  $\mathbf{S}$  is a finite collection of relation symbols  $R$  with specified arity  $\text{arity}(R) \geq 0$ . An  *$\mathbf{S}$ -instance*  $I$  is a finite set of facts, where a fact is an expression of the form  $R(a_1, \dots, a_n)$  with  $R \in \mathbf{S}$  and  $n = \text{arity}(R)$ . Unless specified otherwise, instances are always assumed to be finite. The *active domain*  $\text{adom}(I)$  of  $I$  is the set of all values  $a_i$  occurring in the facts of  $I$ . A *homomorphism*  $h : I \rightarrow J$ , where  $I$  and  $J$  are instances over the same schema  $\mathbf{S}$ , is a function from  $\text{adom}(I)$  to  $\text{adom}(J)$  such that the  $h$ -image of every fact of  $I$  is a fact of  $J$ . We will denote by  $\text{Inst}[\mathbf{S}]$  the set of all  $\mathbf{S}$ -instances.

A  *$k$ -ary pointed  $\mathbf{S}$ -instance* (for  $k \geq 0$ ) is a pair  $(I, \mathbf{a})$  where  $I$  is an  $\mathbf{S}$ -instance and  $\mathbf{a}$  a  $k$ -tuple of elements of  $\text{adom}(I)$ , called *distinguished elements*. A homomorphism  $h : (I, \mathbf{a}) \rightarrow (J, \mathbf{b})$  is a homomorphism  $h : I \rightarrow J$  such that  $h(\mathbf{a}) = \mathbf{b}$ .

**Incidence Graph, Connectedness, C-Acyclicity.** The *incidence graph* of an instance  $I$  is the bipartite multi-graph whose nodes are the elements and the facts of  $I$ , and where there is a distinct (undirected) edge  $(a, f)$  for every occurrence of the element  $a$  in the fact  $f$ . We say that an instance is *connected* if its incidence graph is connected, and an instance is *acyclic* if its incidence graph is acyclic. A pointed instance  $(I, \mathbf{a})$  is *c-acyclic* if every cycle in the incidence graph of  $I$  contains at least one element from the tuple  $\mathbf{a}$ .

**Conjunctive Queries and Unions of Conjunctive Queries.** For  $\mathbf{S}$  a schema and  $k \geq 0$ , a  *$k$ -ary conjunctive query (CQ) over  $\mathbf{S}$*  is an expression of the form

$$q(y_1, \dots, y_k) :- \exists \mathbf{x} (\phi_1 \wedge \dots \wedge \phi_n) \quad (\text{Eq. 1})$$

where each  $\phi_i$  is a relational atomic formula, and such that each variable  $y_i$  occurs in at least one conjunct  $\phi_j$ . A  *$k$ -ary union of conjunctive queries (UCQ) over  $\mathbf{S}$*  is a finite disjunction of  $k$ -ary CQs over  $\mathbf{S}$ . We denote by  $q(I)$  the set of tuples  $\mathbf{a}$  for which it holds that  $I \models q(\mathbf{a})$ .

The *canonical instance* of a CQ of the form (Eq. 1) is the pointed instance  $(I, \mathbf{y})$  where  $I$  is the instance with active domain  $\{y_1, \dots, y_k, \mathbf{x}\}$  whose facts are the conjuncts of  $\phi$ , and  $\mathbf{y} = y_1 \dots y_k$ . Conversely, the *canonical CQ* of a pointed instance  $(I, \mathbf{a})$  with  $\mathbf{a} = a_1 \dots a_k$ , is obtained by associating a unique variable  $y_a$  to each  $a \in \text{adom}(I)$ , letting  $\mathbf{x}$  be an enumeration of all variables  $y_a$  for  $a \in \text{adom}(I) \setminus \{a_1, \dots, a_k\}$ , and taking the query  $q(y_{a_1}, \dots, y_{a_k}) :- \exists \mathbf{x} \bigwedge_{R(b_1, \dots, b_n) \in I} R(y_{b_1}, \dots, y_{b_n})$ . By the well-known Chandra-Merlin theorem, a tuple  $\mathbf{a}$  belongs to  $q(I)$  if and only if the canonical instance of  $q$  homomorphically maps to  $(I, \mathbf{a})$ .

We call a UCQ  $q$  *c-acyclic* if the (pointed) canonical instance of each CQ in  $q$  is c-acyclic.

**Datalog.** A Datalog program is specified by a collection of rules, and it defines a mapping from instances over a schema  $\mathbf{S}_{in}$  (traditionally known as the EDB schema) to instances over a schema  $\mathbf{S}_{out}$  (traditionally known as the IDB schema). The presentation we will give here also allows for auxiliary IDB relations that are not exposed in the output schema.

## 10:4 Right-Adjoints for Datalog Programs

► **Definition 2.1** (Datalog Program). A Datalog program is a tuple  $P = (\mathbf{S}_{in}, \mathbf{S}_{out}, \mathbf{S}_{aux}, \Sigma)$  where  $\mathbf{S}_{in}, \mathbf{S}_{out}, \mathbf{S}_{aux}$  are mutually disjoint schemas, and  $\Sigma$  is a set of rules of the form

$$S(\mathbf{x}) :- R_1(\mathbf{y}_1), \dots, R_n(\mathbf{y}_n)$$

where  $S \in \mathbf{S}_{out} \cup \mathbf{S}_{aux}$ , each  $R_i \in \mathbf{S}_{in} \cup \mathbf{S}_{aux}$ , and each variable in  $\mathbf{x}$  occurs in  $\mathbf{y}_i$  for some  $i$ .

If  $P$  is a Datalog program, then we will often use the notation  $\mathbf{S}_{in}^P, \mathbf{S}_{out}^P, \mathbf{S}_{aux}^P$ , and  $\Sigma^P$  to refer to the constituents of the tuple  $P$ .

The *head* of a rule is the part to the left of the  $:-$  sign, and the *body* is the part to the right. The *canonical instance* of a Datalog rule  $R_0(\mathbf{x}_0) :- R_1(\mathbf{x}_1), \dots, R_n(\mathbf{x}_n)$  is the pointed instance whose active domain is  $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ , whose facts are the conjuncts  $R_i(\mathbf{x}_i)$  of the rule body, and whose sequence of distinguished elements is the tuple  $\mathbf{x}_0$ . We say that a Datalog program  $P$  is *connected* if the canonical instance of each rule is connected. We say that a Datalog program  $P$  is *non-recursive* if  $\mathbf{S}_{aux}^P = \emptyset$ .

If  $P$  is a Datalog program and  $I$  an  $\mathbf{S}_{in}^P$ -instance, then a *solution* for  $I$  with respect to  $P$  is an instance  $J$  over the schema  $\mathbf{S}_{in} \cup \mathbf{S}_{out} \cup \mathbf{S}_{aux}$  such that  $I \subseteq J$ , and such that all the rules of  $P$  are satisfied in  $J$  (i.e., whenever the body of a rule is satisfied under a variable assignment, then so is the head). The well-known *chase* procedure provides a method for constructing a solution: given a Datalog program  $P$  and an  $\mathbf{S}_{in}^P$ -instance  $I$ , we denote by  $\text{chase}_P(I)$  the  $\mathbf{S}_{in}^P \cup \mathbf{S}_{out}^P \cup \mathbf{S}_{aux}^P$ -instance obtained from  $I$  by applying all rules until convergence. More precisely,  $\text{chase}_P(I)$  can be defined as the infinite union  $\bigcup_{i \geq 0} \text{chase}_P^i(I)$ , where  $\text{chase}_P^0(I) = I$ , and where  $\text{chase}_P^{i+1}(I)$  extends  $\text{chase}_P^i(I)$  with all facts that can be derived from facts in  $\text{chase}_P^i(I)$  using a rule in  $\Sigma^P$ . We refer to [1] for more details.

► **Lemma 2.2.** For all Datalog programs  $P$  and  $\mathbf{S}_{in}^P$ -instances  $I$ ,  $\text{chase}_P(I)$  is a solution for  $I$  with respect to  $P$ . Moreover, it is the intersection of all solutions for  $I$  with respect to  $P$ .

We denote the  $\mathbf{S}_{out}^P$ -reduct of  $\text{chase}_P(I)$  by  $P(I)$ . We say that two Datalog programs  $P, P'$  with  $\mathbf{S}_{in}^P = \mathbf{S}_{in}^{P'}$  and  $\mathbf{S}_{out}^P = \mathbf{S}_{out}^{P'}$  are *equivalent* if, for all  $\mathbf{S}_{in}^P$ -instances  $I$ ,  $P(I) = P'(I)$ .

By a *Boolean* Datalog program, we mean a Datalog program  $P$  where  $\mathbf{S}_{out}^P$  consists of a single zero-ary relation symbol, which is customarily denoted as **Ans**. In such cases, write  $P(I) = \text{true}$  if  $P(I) = \{\text{Ans}()\}$  and  $P(I) = \text{false}$  otherwise (i.e., if  $P(I) = \emptyset$ ).

It is well-known that Datalog programs are monotone with respect to homomorphisms:

► **Lemma 2.3.** Let  $P$  be any Datalog program, and let  $I, I'$  be  $\mathbf{S}_{in}^P$ -instances. Every homomorphism  $h : I \rightarrow I'$  yields, when restricted to  $\text{adom}(P(I))$ , a homomorphism from  $P(I)$  to  $P(I')$ .

We can think of the above definition of  $P(I)$ , in terms of the chase, as a bottom-up account of the semantics of a Datalog program. *Unfoldings* (a.k.a. *expansions*) provide a complementary, top-down account. Given a Datalog program  $P$ , the set of *derivable rules* of  $P$  is the smallest set of rules that (i) contains all rules of  $P$ , and (ii) is closed under the operation of substituting occurrences of rule heads by the corresponding rule bodies (renaming variables as necessary). Given a Datalog program  $P$  and a relation  $R \in \mathbf{S}_{out}^P$ ,  $\text{Unfoldings}(P, R)$  is the set of canonical instances of derivable rules that have  $R$  in the rule head and that only have  $\mathbf{S}_{in}$ -relations in the body. Note that this set is in general infinite.

► **Example 2.4.** Let  $P$  be the Datalog program consisting of the three rules

$$R(x, y) :- S(x, y) \qquad R(x, x) :- T(x, y) \qquad T(x, y) :- U(x, y), U(y, z)$$

where  $\mathbf{S}_{in} = \{U, S\}$ ,  $\mathbf{S}_{out} = \{R\}$ , and  $\mathbf{S}_{aux} = \{T\}$ . Then the derivable rules of  $P$  are the rules of  $P$  together with the rule  $R(x, x) :- U(x, y), U(y, x)$ , and  $\text{Unfoldings}(P, R)$  consists (up to isomorphism) of the pointed instances  $(\{U(a, b), U(b, c)\}, \langle a, a \rangle)$  and  $(\{S(a, b)\}, \langle a, b \rangle)$ .

► **Lemma 2.5** (Cf. [12]). *For all Datalog programs  $P$ , instances  $I \in \text{Inst}[\mathbf{S}_{in}^P]$ , and  $\mathbf{S}_{out}^P$ -facts  $R(\mathbf{a})$  over  $\text{adom}(I)$ ,  $R(\mathbf{a}) \in P(I)$  iff, for some  $(J, \mathbf{b}) \in \text{Unfoldings}(P, R)$ ,  $(J, \mathbf{b}) \rightarrow (I, \mathbf{a})$ .*

### 3 TAM Datalog

TAM Datalog is a fragment of Datalog defined by two requirements: “tree-shaped” and “almost-monadic”. We introduce each in isolation first.

**Almost-Monadic Datalog.** Recall that a Datalog program is *monadic* if all relations in  $\mathbf{S}_{aux}$  are unary. It is well known that monadic Datalog programs can be expressed in Monadic Second-Order logic (MSO). Formally, by a  $k$ -ary MSO query over a schema  $\mathbf{S}$ , we will mean an MSO formula  $\phi(x_1, \dots, x_k)$  over  $\mathbf{S}$ . We say that a Datalog program  $P = (\mathbf{S}_{in}, \mathbf{S}_{out}, \mathbf{S}_{aux}, \Sigma)$  together with a designated  $k$ -ary relation  $R \in \mathbf{S}_{out}$ , defines an MSO query  $\phi_R(\mathbf{x})$  over  $\mathbf{S}_{in}$ , if for all  $\mathbf{S}_{in}$ -instances  $I$  and  $\mathbf{a} \in \text{adom}(I)$ ,  $R(\mathbf{a}) \in P(I)$  iff  $I \models \phi_R(\mathbf{a})$ . The following is folklore in the database literature (cf. [19] for an explicit proof):

► **Theorem 3.1.** *Let  $P$  be a monadic Datalog program and  $R \in \mathbf{S}_{out}^P$ . Then  $(P, R)$  defines an MSO query.*

We will now define a weaker restriction, namely that of *almost-monadic* Datalog programs, for which the same holds. These are programs in which every  $k$ -ary auxiliary relation has, among its  $k$  argument positions, (at most) one specified “*articulation position*”, and the syntax of the rules is constrained in such a way that variables occurring in non-articulation positions can only be used to carry information forward, and not to “perform joins”. Formally:

► **Definition 3.2** (Almost-Monadic Datalog). *An articulation function, for a Datalog program  $P$ , is a partial function  $f$  mapping relations  $R \in \mathbf{S}_{aux}^P$  to a number  $f(R) \in \{1, \dots, \text{arity}(R)\}$ , which we will call the *articulation position* of  $R$ . Each  $i \in \{1, \dots, \text{arity}(R)\}$  other than  $f(R)$  is called a *non-articulation position* of  $R$ . A Datalog program is *almost-monadic* if there exists an articulation function such that, in every rule, each variable occurring in a non-articulation position of an auxiliary relation in a rule body occurs only once in that rule body, and does not occur in the articulation position of an auxiliary relation in the head.*

Note: the articulation conditions pertain to auxiliary relations and not to output relations.

► **Example 3.3.** The Datalog program from Example 1.1 (which outputs all pairs  $(a, b)$  for which there is a directed path from  $a$  to  $b$ ) is not monadic but is almost-monadic. The witnessing articulation function assigns to the auxiliary relation  $\text{Path}$  its first position as articulation position. Even if we extend the program with an additional rule  $\text{Ans}(x, y) :- \text{Path}(y, x)$  (so that it computes all pairs  $(a, b)$  for which there is a directed path from  $a$  to  $b$  or from  $b$  to  $a$ ), the resulting program is still almost-monadic. This is because the requirements on the articulation function only pertain to auxiliary relations, not to output relations. On the other hand, if we were to change the rule  $\text{Path}(x, y) :- \text{Edge}(x, z), \text{Path}(z, y)$  to  $\text{Path}(x, y) :- \text{Path}(x, z), \text{Path}(z, y)$ , the program would no longer be almost-monadic (cf. also Example 6.2).

For an example of a Datalog program that is *not* almost-monadic, see Example 3.11 below.

## 10:6 Right-Adjoints for Datalog Programs

► **Proposition 3.4.** *The almost-monadic Datalog program from Example 1.1 is not equivalent to a monadic Datalog program.*

The following result justifies the terminology *almost-monadic*. It shows that almost-monadic Datalog programs can be simulated, in a precise sense, by monadic Datalog programs.

► **Theorem 3.5.** *For each almost-monadic Datalog program  $P$  and  $k$ -ary relation symbol  $R \in \mathbf{S}_{out}^P$ , there is a Boolean monadic Datalog program  $P'$  where  $\mathbf{S}_{in}^{P'} = \mathbf{S}_{in}^P \cup \{Q_1, \dots, Q_k\}$ , such that the following are equivalent, for all  $\mathbf{S}_{in}^P$ -instances  $I$  and  $a_1, \dots, a_k \in \text{adom}(I)$ :*

1.  $R(a_1, \dots, a_k) \in P(I)$ ,
2.  $P'(I \cup \{Q_1(a_1), \dots, Q_k(a_k)\}) = \text{true}$ .

► **Example 3.6.** Let  $P$  be the Datalog program from Example 1.1. To satisfy the statement of Theorem 3.5, it suffices to define  $P'$  as:

$\text{Path}'(x) :- \text{Edge}(x, y), Q_2(y)$ .

$\text{Path}'(x) :- \text{Edge}(x, y), \text{Path}'(y)$ .

$\text{Ans}() :- \text{Path}'(x), Q_1(x)$ .

Informally,  $\text{Path}'(x)$  holds if there is a path starting at  $x$  that ends at a node satisfying  $Q_2$ .

It follows that almost-monadic Datalog is contained in MSO. That is, we have the following analogue of Thm. 3.1 for almost-monadic Datalog programs:

► **Corollary 3.7.** *Let  $P$  be an almost-monadic Datalog program and  $R \in \mathbf{S}_{out}^P$ . Then  $(P, R)$  defines an MSO query.*

In summary, we have that almost-monadic Datalog forms a strict extension of monadic Datalog that is still contained in MSO. One may be tempted to conjecture that almost-monadic Datalog is expressively complete for the intersection of Datalog and MSO. However, this is not the case. The easiest way to show this, is using the following Lemma, which is interesting in its own right, as it shows that, when the output schema contains only unary relations, almost-monadic Datalog is no more expressive than monadic Datalog:

► **Lemma 3.8.** *Let  $P$  be any almost-monadic Datalog such that every  $R \in \mathbf{S}_{out}^P$  is unary. Then  $P$  is equivalent to a monadic Datalog program.*

Using this lemma, we can show:

► **Proposition 3.9.** *The unary MSO query “ $x$  lies on a directed  $R$ -cycle” is not definable by an almost-monadic Datalog program.*

► **Remark 3.10.** Prop. 3.9 also shows that almost-monadic Datalog is not closed under composition, because the same query can be expressed as the composition of two almost-monadic Datalog programs, where the first computes the transitive closure  $R^*$  of the relation  $R$ , and the second program consists of the single non-recursive rule  $\text{Ans}(x) :- R^*(x, x)$ . It also shows that almost-monadic Datalog is strictly included in MODEQ (also known as Flag-and-Check), which is another language contained in the intersection of Datalog and MSO [23]. See also [7] for a characterization of the intersection of MSO and Datalog in terms of infinite domain constraint satisfaction problems. As we will soon see, the intersection of *tree-shaped* Datalog and MSO is (up to logical equivalence) precisely tree-shaped almost-monadic Datalog.



**Tree-shapedness.** We say that a Datalog program  $P$  is *tree-shaped* if the incidence graph of the canonical instance of each rule is acyclic. In particular, since we defined incidence graphs as multigraphs, this implies that no variable occurs twice in the same conjunct in the rule body (but the rule head may contain repeated occurrences of variables). Note that we do not require the incidence graph of the rules to be connected, nor do we make any requirements (say, in the case of binary relations) on the direction of edges. Thus, in tree-shaped Datalog programs, rules such as  $T(x) :- R(x, y), S(x, y)$  or  $T(x) :- R(x, x)$  are forbidden.

► **Example 3.11.** Consider the tree-shaped Datalog program  $P$  given by the following two rules (where  $\mathbf{S}_{in}^P$  consists of two binary relations,  $E, F$ ):

$$R(x, y) :- E(x, u), F(u, y) \quad R(x, y) :- E(x, u), R(u, v), F(v, y) \quad \mathbf{Ans}(x, y) :- R(x, y)$$

Then  $\mathbf{Ans}^{P(I)}$  contains all pairs  $(a, b)$ , such that there is a directed path from  $a$  to  $b$  in  $I$  consisting of a number of  $E$ -edges followed by an equal number of  $F$ -edges.

Observe that  $P$  is not TAM Datalog because neither the first position of the relation  $R$  qualifies as an articulation position (since  $v$  occurs twice in the second rule body) nor the second position (since  $u$  occurs twice in the same rule body).

It follows from known facts about MSO (viz. the fact that MSO on words captures the regular languages) that  $(P, \mathbf{Ans})$  does not define an MSO query. In particular,  $P$  is not equivalent to a monadic Datalog program, or even an almost-monadic Datalog program.

► **Lemma 3.12.** *Let  $P$  be any tree-shaped Datalog program. Then, for each  $R \in \mathbf{S}_{out}^P$ ,  $\text{Unfoldings}(P, R)$  consists of acyclic pointed instances.*

**TAM Datalog.** A *TAM Datalog program* is a tree-shaped, almost-monadic Datalog program. We will give a precise model-theoretic characterization of TAM Datalog in terms of MSO.

We say that an MSO query  $\phi(\mathbf{x})$  is *tree-determined* if for each pointed instance  $(I, \mathbf{a})$ , we have that  $I \models \phi(\mathbf{a})$  if and only if there is an acyclic pointed instance  $(J, \mathbf{b})$  such that  $J \models \phi(\mathbf{b})$  and  $(J, \mathbf{b}) \rightarrow (I, \mathbf{a})$ . Note that  $J$  must be finite and that  $J$  is not required to be connected.

► **Theorem 3.13.** *Let  $\phi(x_1, \dots, x_n)$  be an MSO formula. The following are equivalent:*

1.  $\phi$  is definable by a TAM Datalog program,
2.  $\phi$  is definable by a tree-shaped Datalog program,
3.  $\phi$  is tree-determined.

► **Remark 3.14.** It is worth comparing this to the result in [19] that states that monadic Datalog and MSO have the same expressive power on finite trees. Besides the fact that Thm. 3.13 is a characterization on arbitrary (finite) instances while the result in [19] is restricted to trees, there are a few other important differences: in [19], it is assumed that trees are represented as structures in which the children of each node are ordered; that the signature includes predicates marking the root, leafs, the first child of each node, and the last child of each node; and that each node of the tree is labeled by precisely one of the (other) unary predicates in the signature. These assumptions together imply that every homomorphism between such trees is necessarily an isomorphism, which makes the two results incomparable.

► **Corollary 3.15** (TAM Datalog is closed under composition). *For all TAM Datalog programs  $P_1$  and  $P_2$  with  $\mathbf{S}_{in}^{P_2} = \mathbf{S}_{out}^{P_1}$ , there is a TAM Datalog program  $P_3 = (\mathbf{S}_{in}^{P_1}, \mathbf{S}_{out}^{P_2}, \mathbf{S}'_{aux}, \Sigma')$  such that, for all  $\mathbf{S}_{in}^{P_1}$ -instances  $I$ ,  $P_3(I) = P_2(P_1(I))$ .*

We also provide a syntactic normal form for TAM Datalog programs. A TAM Datalog program is *simple* if every rule body contains precisely one occurrence of a relation from  $\mathbf{S}_{in}$ . For instance the program given in Example 1.1 is a simple TAM Datalog program.

► **Theorem 3.16.** *Every (connected) TAM Datalog program can be transformed in polynomial-time into an equivalent (connected) simple TAM Datalog program.*

We make use of this normal form in some of our proofs.

## 4 Right-Adjoints for TAM Datalog

The notion of adjunction comes from category theory. Although for the most part, we do not assume that the reader has a background in category theory, in order to motivate our definition of generalized right-adjoints for Datalog programs, it is helpful to briefly discuss Datalog programs from a categorical perspective.

Recall that each Datalog program  $P$  defines a mapping from  $\text{Inst}[\mathbf{S}_{in}^P]$  to  $\text{Inst}[\mathbf{S}_{out}^P]$ , where  $\text{Inst}[\mathbf{S}]$  denotes the set of all  $\mathbf{S}$ -instances. Recall also that this mapping is monotone with respect to homomorphisms (cf. Lemma 2.3). We view  $\text{Inst}[\mathbf{S}_{in}^P]$  and  $\text{Inst}[\mathbf{S}_{out}^P]$  as partial (pre)orders, which can consequently be viewed as *thin categories* where the objects are the  $\mathbf{S}$ -instances and there is an arrow from  $I$  to  $J$  if there exists a homomorphism  $h : I \rightarrow J$ . The categorical notion of a *functor* is then simply a monotone mapping. In particular, each Datalog program  $P$  defines a functor. For functors  $F : X \rightarrow Y$  and  $G : Y \rightarrow X$ , where  $X$  and  $Y$  are arbitrary thin categories, it is said that  $G$  is a *right-adjoint* for  $F$ , and that  $F$  is a *left-adjoint* of  $G$ , if it holds that  $F(I) \rightarrow J$  iff  $I \rightarrow G(J)$ .<sup>1</sup>

In this section, we study the existence of right-adjoints for Datalog programs.

► **Example 4.1.** Consider the Datalog program  $P = (\mathbf{S}_{in}, \mathbf{S}_{out}, \emptyset, \Sigma)$ , where  $\mathbf{S}_{in} = \{R\}$ ,  $\mathbf{S}_{out} = \{S\}$ , and  $\Sigma$  consists of the rules  $S(x, y) :- R(x, y)$  and  $S(x, y) :- R(y, x)$ . If we think of an input instance  $I$  as a directed graph,  $P(I)$  is its *symmetric closure*. For every  $\mathbf{S}_{out}^P$ -instance  $J$ , let  $\Omega(J)$  be the  $\mathbf{S}_{in}^P$ -instance that is the *maximal symmetric sub-instance* of  $J$ , that is,  $\Omega(J)$  consists of all facts  $R(x, y)$  for which it holds that  $J$  contains both  $S(x, y)$  and  $S(y, x)$ . It is not hard to see that  $P(I) \rightarrow J$  iff  $I \rightarrow \Omega(J)$ . Hence,  $\Omega$  is a right-adjoint of  $P$ .

► **Example 4.2.** Consider the TAM Datalog program  $P = (\mathbf{S}_{in}, \mathbf{S}_{out}, \emptyset, \Sigma)$ , where  $\mathbf{S}_{in} = \{Q_1, Q_2\}$ ,  $\mathbf{S}_{out} = \{Q_3\}$ , and  $\Sigma$  consists of the rule  $Q_3() :- Q_1(x), Q_2(y)$ . This Datalog program does *not* have a right-adjoint in the above sense. Indeed, let  $J$  be the empty instance. Then  $P(I) \rightarrow J$  holds if and only if either  $I$  has no  $Q_1$ -facts or  $I$  has no  $Q_2$ -facts, a condition that cannot be equivalently characterized by the existence of a homomorphism from  $I$  to any fixed single instance  $J'$ . However, it can be shown that  $P(I) \rightarrow J$  if and only if either  $I \rightarrow J'_1$  or  $I \rightarrow J'_2$ , where  $J'_1 = \{Q_1(a)\}$  and  $J'_2 = \{Q_2(a)\}$ . If we generalize the notion of right-adjoint by allowing  $\Omega(J)$  to be a finite set of instances, then, as we will see later (Thm. 4.5),  $P$  *does* admit such a right-adjoint, and the fact that  $\Omega(J)$  needs to consist of multiple instances is related to the fact that the program is not connected.

Motivated by the above examples and other considerations that will become clear soon, the precise notion of right-adjoints that we will adopt here is a little more refined:

<sup>1</sup> Note that this coincides with the usual definition of adjoint functors as long as both categories are thin. It is also precisely the notion of right-adjoints for Galois connections over preordered sets. See also Remark 4.11 for why we adopt this “thin” definition of adjoints.

► **Definition 4.3** (Generalized Right-Adjoints). *A generalized right-adjoint for a Datalog program  $P$  is a function  $\Omega_P$  that maps each  $J \in \text{Inst}[\mathbf{S}_{out}^P]$  to a finite set of pairs  $(J', \iota)$  where  $J' \in \text{Inst}[\mathbf{S}_{in}^P]$  and  $\iota : J' \rightarrow J$  is a partial function, such that the following holds: for all  $I \in \text{Inst}[\mathbf{S}_{in}^P]$ , there is a homomorphism  $h : P(I) \rightarrow J$  iff there is a homomorphism  $h' : I \rightarrow J'$  for some  $(J', \iota) \in \Omega_P(J)$ , and, furthermore, the homomorphism  $h'$ , respectively  $h$ , can be chosen such that the following diagram commutes.<sup>2</sup>*

$$\begin{array}{ccc} \text{adom}(P(I)) & \xrightarrow{h} & \text{adom}(J) \\ \text{id} \uparrow & & \uparrow \iota \\ \text{adom}(I) & \xrightarrow{h'} & \text{adom}(J') \end{array}$$

Here, the “ $\rightarrow$ ” arrow refers to homomorphisms, and “ $\rightarrow$ ” is used to refer to a partial function. The partial functions  $\iota$  are needed later on to reason about pointed instances (cf. for instance the proof of Theorem 5.4).

This notion of generalized right-adjoint behaves as one would expect. In particular, if two Datalog programs have generalized right-adjoints, then so does their composition.

► **Example 4.4.** Let  $P$  be the Datalog program with input schema  $\{R_1, R_2\}$  and output schema  $\{Q_1, Q_2\}$  and consisting of the rules

$$Q_1(x) :- R_1(x) \qquad Q_1(x) :- R_2(x) \qquad Q_2(x) :- R_1(x), R_2(x)$$

This Datalog program indeed has a generalized right-adjoint  $\Omega_P$ . In fact it has a regular right-adjoint, in the sense that  $\Omega_P(J)$  is a singleton for all  $J \in \text{Inst}[\mathbf{S}_{out}^P]$ , as will follow from Theorem 4.5 below. We will illustrate this with an example. Let  $J$  be the  $\mathbf{S}_{out}^P$ -instance consisting of the single fact  $Q_1(a)$ . Then a suitable choice for  $\Omega_P(J)$  is the singleton set consisting of the pair  $(J', \iota)$ , where  $J'$  consists of the facts  $R_1(a_1), R_2(a_2)$  and  $\iota(a_1) = \iota(a_2) = a$ . Indeed, for each  $\mathbf{S}_{in}^P$ -instance  $I$ ,  $P(I) \rightarrow J$  iff  $I \rightarrow J'$  via homomorphisms that make the diagram in Definition 4.3 commute. Note that, to make the diagram commute,  $J'$  must indeed contain facts of the form  $R_1(a_1)$  and  $R_2(a_2)$  for distinct values  $a_1, a_2$  that are both mapped to  $a$  by  $\iota$ .

Our main result in this section is:

► **Theorem 4.5.** *Every TAM Datalog program  $P$  has a generalized right-adjoint  $\Omega_P$ . If  $P$  is connected, then  $\Omega_P(J)$  is a singleton for all  $J \in \text{Inst}[\mathbf{S}_{out}^P]$ . Moreover,  $\Omega_P(J)$  is computable from  $J$  and  $P$  in  $2\text{ExpTime}$ , and in  $\text{ExpTime}$  whenever the arity of  $P$  is bounded.*

**Proof.** We first consider the special case of connected programs. We may assume without loss of generality that  $P$  is simple. This means that every rule is of the following form:

$$R_0(\mathbf{x}_0) :- E(\mathbf{y}), R_1(\mathbf{x}_1), \dots, R_m(\mathbf{x}_m) \tag{Eq. 2}$$

$$R(\mathbf{x}) :- E(\mathbf{y}), R_1(\mathbf{x}_1), \dots, R_m(\mathbf{x}_m) \tag{Eq. 3}$$

where  $E$  is an input relation, each  $R_i$  is an auxiliary relation, and  $R$  is an output relation.

To simplify the exposition below, we introduce some further notation. For each atom  $R_i(\mathbf{x}_i)$  as in the above rule types, we will denote by  $p_i \in \{1, \dots, n\}$  (with  $n = \text{arity}(E)$ ) the unique number such that  $y_{p_i}$  is equal to the articulated variable in  $R_i(\mathbf{x}_i)$ . It indeed follows from the definition of TAM Datalog and the assumed connectedness and simplicity of  $P$  that such an index exists and is unique.

<sup>2</sup> By this we, we mean that, for all  $a \in \text{adom}(I)$ , either  $h(a) = \iota(h'(a))$  or both are undefined.

## 10:10 Right-Adjoins for Datalog Programs

We construct an  $\mathbf{S}_{in}$ -instance  $J'$  consisting of all facts  $E((b_1, X_1), \dots, (b_n, X_n))$  where

1. Each  $b_i$  is an element of  $\text{adom}(J) \cup \{\perp\}$  and  $X_i$  is a set of  $\mathbf{S}_{aux}$ -facts over  $\text{adom}(J) \cup \{\perp\}$  (not necessarily facts of  $J$ ) in which  $b_i$  occurs in articulation position;
2. For each rule of the form (1) above and for each map  $g : \{\mathbf{y}, \mathbf{x}_1, \dots, \mathbf{x}_m\} \rightarrow \text{adom}(J) \cup \{\perp\}$ , if for each  $1 \leq i \leq m$ ,  $R_i(g(\mathbf{x}_i)) \in X_{p_i}$  then  $R_0(g(\mathbf{x}_0)) \in X_{p_0}$ ; and
3. For each rule of the form (2) above and for each map  $g : \{\mathbf{y}, \mathbf{x}_1, \dots, \mathbf{x}_m\} \rightarrow \text{adom}(J) \cup \{\perp\}$ , if for each  $1 \leq i \leq m$ ,  $R_i(g(\mathbf{x}_i)) \in X_{p_i}$  then  $R(g(\mathbf{x}))$  is a fact of  $J$ .

Note that the total number of possible facts  $E((b_1, X_1), \dots, (b_n, X_n))$  in  $J'$  is double exponential in the combined size of  $P$  and  $J$ , and is exponential in  $J$  if the arity of  $P$  is bounded.

Let  $\iota$  be the natural projection from  $J'$  to  $J$ , mapping all elements of the form  $(a, X)$  to  $a$  (and undefined on elements of the form  $(\perp, X)$ ).

▷ **Claim.** For all  $\mathbf{S}_{in}$ -instances  $I$ ,  $P(I) \rightarrow J$  iff  $I \rightarrow J'$ . Moreover, the witnessing homomorphisms can be constructed so that the following diagram commutes:

$$\begin{array}{ccc} P(I) & \longrightarrow & J \\ id \uparrow & & \uparrow \iota \\ I & \longrightarrow & J' \end{array}$$

**Proof.** [ $\Rightarrow$ ] Let  $h : P(I) \rightarrow J$ . Recall that we denote by  $\text{chase}_P(I)$  the  $\mathbf{S}_{in} \cup \mathbf{S}_{out} \cup \mathbf{S}_{aux}$ -instance that is the chase of  $I$  (and of which  $I$  and  $P(I)$  are the  $\mathbf{S}_{in}$ -reduct and  $\mathbf{S}_{out}$ -reduct, respectively). We extend  $h$  to the entire active domain of  $\text{chase}_P(I)$  by sending every element  $a$  that is not in  $\text{adom}(P(I))$  to a fresh value  $\perp$ . With a slight abuse of notation, in what follows, we denote by  $h$  the extended map from  $\text{adom}(\text{chase}_P(I))$  to  $\text{adom}(J) \cup \{\perp\}$ . For each  $a \in \text{adom}(\text{chase}_P(I))$ , let  $F_a$  be the set of all  $\mathbf{S}_{aux}$ -facts of  $\text{chase}_P(I)$  in which  $a$  occurs in articulation position. We define  $h'(a) = (h(a), h(F_a))$ .

We claim that  $h'$  is a homomorphism from  $I$  to  $J'$ . Let  $E(a_1, \dots, a_n)$  be any fact of  $I$ . We must show that the fact  $E((h(a_1), h(F_{a_1})), \dots, (h(a_n), h(F_{a_n})))$  belongs to  $J'$ . That is, we must show that conditions 1–3 hold.

Clearly, the first requirement is satisfied, namely,  $h(X_{a_i})$  consists of facts in which  $h(a_i)$  occurs in articulation position.

To see that the second requirement holds, consider a rule of form (1) and any map  $g : \{\mathbf{y}, \mathbf{x}_1, \dots, \mathbf{x}_m\} \rightarrow \text{adom}(J)$ , such that, for each  $1 \leq i \leq m$ ,  $R_i(g(\mathbf{x}_i)) \in X_{p_i}$ . By construction, this means that each fact  $R_i(g(\mathbf{x}_i))$  is the  $h$ -image of a fact  $R_i(\mathbf{b}_i)$  in  $\text{chase}_P(I)$ . Now consider the map  $\tilde{g} : \{\mathbf{y}, \mathbf{x}_1, \dots, \mathbf{x}_m\} \rightarrow \text{adom}(I)$  defined by  $\tilde{g}(\mathbf{y}) = (a_1, \dots, a_n)$  and  $\tilde{g}(\mathbf{x}_i) = \mathbf{b}_i$  for  $1 \leq i \leq m$ . Note that  $\tilde{g}$  is a well-defined function. Indeed, for every  $1 \leq i \leq m$  and every  $z \in R(\mathbf{x}_i)$ , if  $z$  occurs more than once in the rule, then  $z$  must necessarily be the variable that appears in the articulation position  $p_i$  of  $R(\mathbf{x}_i)$ . It follows that  $z$  occurs in  $\mathbf{y}$  at position  $p_i$ , and, hence, necessarily,  $R_i(g(\mathbf{x}_i))$  belongs to the  $h$ -image of  $X_{a_{p_i}}$ , and, hence,  $\tilde{g}(z) = a_{p_i}$ .

Since  $\text{chase}_P(I)$  is closed under the rules of  $P$ , we may conclude that  $R_0(\tilde{g}(\mathbf{x}_0))$  belongs to  $\text{chase}_P(I)$ . Let  $z$  be the variable occurring in articulation position in  $R_0(\mathbf{x}_0)$ . Recall that  $z$  occurs in  $\mathbf{y}$  at position  $p_0$ , and  $\tilde{g}(z) = a_{p_0}$ . Then  $R_0(\tilde{g}(\mathbf{x}_0)) \in F_{a_{p_0}}$ , and hence,  $h(F_{a_{p_0}})$  contains  $R_0(h \circ \tilde{g}(\mathbf{x}_0))$ . Note that by definition,  $h \circ \tilde{g} = g$ . In particular,  $R_0(h \circ \tilde{g}(\mathbf{x}_0)) = R_0(g(\mathbf{x}_0))$ . Therefore we have that  $R_0(g(\mathbf{x}_0)) \in h(F_{a_{p_0}})$ , and we are done.

To see that the third requirement holds, consider a rule of form (2) and any map  $g : \{\mathbf{y}, \mathbf{x}_1, \dots, \mathbf{x}_m\} \rightarrow \text{adom}(J)$ , such that, for each  $1 \leq i \leq m$ ,  $R_i(g(\mathbf{x}_i)) \in X_{p_i}$ . By exactly the same reasoning as before, an  $h$ -preimage of the rule head  $R(g(\mathbf{x}))$  belongs to  $\text{chase}_P(I)$ . Hence, it belongs to  $P(I)$ , therefore,  $R(g(\mathbf{x}))$  is a fact of  $J$ .

It is also clear from the construction that  $h \circ id = \iota \circ h'$ , where  $id$  is the identity function on  $\text{adom}(I) \cap \text{adom}(P(I))$ . That is, the diagram commutes.

[ $\Leftarrow$ ] Conversely, let  $h : I \rightarrow J'$ . Note that  $\text{adom}(\text{chase}_P(I)) = \text{adom}(I)$ , and hence we  $h(a)$  is well-defined for all  $a \in \text{adom}(\text{chase}_P(I))$ . Let  $h' : \text{adom}(I) \rightarrow \text{adom}(J)$  be the map such that  $h'(a) = b$  whenever  $h(a) = (b, X)$ .

▷ **Subclaim 1.** For all  $a \in \text{adom}(\text{chase}_P(I))$ , if  $h(a) = (b, X)$ , then the  $h'$ -image of every  $\mathbf{S}_{aux}$ -fact of  $\text{chase}_P(I)$  in which  $a$  occurs in articulation position belongs to  $X$ .

▷ **Subclaim 2.**  $h'$  is a homomorphism from  $P(I)$  to  $J$ .

Subclaim 1 can be proved by induction on the derivation length of the fact in question.

To prove subclaim 2, let  $R(\mathbf{a})$  be an  $\mathbf{S}_{out}$ -fact belonging to  $P(I)$ . Its derivation must use a rule of the form (2) above, using an assignment  $g$  (where  $g(\mathbf{x}) = \mathbf{a}$ ). By Subclaim 1, we have that  $R_i(h'(g(\mathbf{x}_i)))$  belongs to  $h(g(y_{p_i}))_2$ , for  $y_{p_i}$  the articulated variable in  $\mathbf{x}_i$ . Furthermore,  $E(g(\mathbf{y}))$  holds in  $I$ , and hence  $E(h(g(\mathbf{y})))$  holds in  $J'$ . By construction of  $J'$ , this means that the  $R(h'(g(\mathbf{x})))$ , that is,  $R(h'(\mathbf{a}))$ , belongs to  $J$ . This concludes the proof for the case of *connected* TAM Datalog programs.

It is also clear from the construction that  $\iota \circ h = h' \circ id$ , where  $id$  is the identity function on  $\text{adom}(I) \cap \text{adom}(P(I))$ . That is, the diagram commutes.  $\triangleleft$

Finally, we show how to handle non-connected TAM Datalog programs. Let  $P$  be a non-connected TAM Datalog program. Let  $P'$  be obtained from  $P$  by adding a fresh binary input-relation  $S$ , and using this relation to make every every rule connected in some arbitrary way (more precisely, whenever the incidence graph of a rule body has multiple connected component, we add  $S$ -atoms to the body connecting these components while preserving tree-shapedness and almost-monadicity. For every input instance  $I$ , we denote by  $\hat{I}$  the  $\mathbf{S}_{in} \cup \{S\}$ -instance extending  $I$  with all facts of the form  $S(a, b)$  for  $a, b \in \text{adom}(I)$ . Furthermore, given an instance  $J'$  over the schema  $\mathbf{S}_{in} \cup \{S\}$ , by an “ $S$ -component” of  $J'$  we will mean the  $\mathbf{S}_{in}$ -retract of a fully  $S$ -connected sub-instance of  $J'$ . Clearly, if  $J$  is an  $\mathbf{S}_{in}$ -instance and  $J'$  is a  $\mathbf{S}_{in} \cup \{S\}$ -instance, then  $\hat{J} \rightarrow J'$  iff  $J \rightarrow J''$  for some  $S$ -component  $J''$  of  $J'$ . Now we simply define  $\Omega_P(J)$  to be the set of all  $S$ -components of instance in  $\Omega_{P'}(J)$ . Then we have:  $P(I) \rightarrow J$  iff  $P'(\hat{I}) \rightarrow J$  iff  $\hat{I} \rightarrow \Omega_{P'}(J)$  iff  $I \rightarrow J'$  for some  $J' \in \Omega_P(J)$ .

As a side remark, we mention that there is another way present the final argument where we lift the connected case to the general case: we can view the function that sends  $I$  to  $\hat{I}$  as a functor that itself has a generalized right-adjoint (sending  $I$  to its  $S$ -connected components). Thus, we can argue by composition of adjoints.  $\blacktriangleleft$

We note that the proof of `thm:tam-adjoint` makes crucial use of both the tree-shapedness and the almost-monadicity of the Datalog program. Indeed, both properties are important for the existence of generalized right-adjoints as the following two propositions show:

► **Proposition 4.6.** *The tree-shaped Datalog program  $P$  in Example 3.11 (which is not almost-monadic) does not admit a generalized right-adjoint.*

**Proof.** Assume towards a contradiction that  $P$  has a generalized right-adjoint  $\Omega_P$ . Let  $J$  be the two-element  $\{R\}$ -instance consisting of the facts  $R(0, 1)$  and  $R(1, 0)$ .

For  $n \geq 1$ , let  $C_n$  be the  $\{E, F\}$ -instance consisting of the facts  $E(v_0, v_1), \dots, E(v_{n-1}, v_n), E(v_n, v_0)$ , that is, the directed  $E$ -cycle of length  $n$ . Trivially,  $P(C_n) \rightarrow J$  for all  $n \geq 1$ . Therefore, for each  $n \geq 1$ , we have  $C_n \rightarrow J'$  for some  $J' \in \Omega_P(J)$ . For every  $J' \in \Omega_P(J)$  and for each element  $b$  of  $J'$ , let us define  $n_{J', b}$  to be an arbitrarily chosen value such that

## 10:12 Right-Adjoins for Datalog Programs

$(C_n, v_0) \rightarrow (J', b)$ , or undefined, if no such value exists. It follows from our earlier observation that  $n_{J', b}$  is defined for at least one pair  $(J', b)$  with  $J' \in \Omega_P(J)$ . Let  $m$  be a common multiple of all defined  $n_{J', b}$ 's.

For each pair of positive integers  $e \leq f$ , let  $I_{e, f}$  be the  $\{E, F\}$ -instance depicted as follows:

$$u_0 \xrightarrow{E} v_0 \xrightarrow{F} u_1 \xrightarrow{E} v_1 \xrightarrow{F} u_2 \xrightarrow[\text{sequence of } e \text{ } E\text{-edges}]{\text{sequence of } e \text{ } E\text{-edges}} z \xrightarrow[\text{sequence of } f \text{ } F\text{-edges}]{\text{sequence of } f \text{ } F\text{-edges}} u_0$$

▷ **Claim 1.** For all  $1 \leq e \leq f$ , the following are equivalent:

1.  $I_{e, f} \rightarrow J'$  for some  $J' \in \Omega_P(J)$
2.  $e \neq f$ .

*Proof.* If  $e = f$  then  $P(I_{e, f})$  contains an  $R$ -cycle of odd length, viz.  $u_0 \xrightarrow{R} u_1 \xrightarrow{R} u_2 \xrightarrow{R} u_0$ , and therefore  $P(I_{e, f}) \not\rightarrow J$ . Hence,  $I_{e, f} \not\rightarrow J'$  for all  $J' \in \Omega_P(J)$ . On the other hand, if  $e < f$ , then  $P(I_{e, f})$  is a disjoint union of  $R$ -paths, and, clearly,  $P(I_{e, f}) \rightarrow J$ . Therefore,  $I_{e, f} \rightarrow J'$  for some  $J' \in \Omega_P(J)$ . ◀

Now, let  $e$  be larger than the universe of all instances in  $\Omega_P(J)$  and let  $f = e + m$ . By Claim 1, there is a homomorphism  $h : I_{e, f} \rightarrow J'$  for some  $J' \in \Omega_P(J)$ . We will show that  $h$  can be extended to a homomorphism  $h' : I_{f, f} \rightarrow J'$ , which contradicts Claim 1. Let

$$u_2 = x_0 \xrightarrow{E} x_1 \cdots \xrightarrow{E} x_e = z$$

be the sub-instance of  $I_{e, f}$  consisting of the  $E$ -edges joining  $u_2$  and  $z$ . Similarly, let

$$u_2 = x_0 \xrightarrow{E} x_1 \cdots \xrightarrow{E} x_e \xrightarrow{E} x_{e+1} \cdots \xrightarrow{E} x_f = z$$

be the sub-instance of  $I_{f, f}$  consisting of the  $E$ -edges joining  $u_2$  and  $z$ . Recall that  $f = e + m$ . Since  $e$  is larger than the domain size of  $J'$ , it must be the case that  $h(x_i) = h(x_j) = b$  for some  $i < j \leq e$ , and for some element  $b$  of  $J'$ . This means that  $b$  lies on a directed  $E$ -cycle in  $J'$ , and hence, in particular, it lies on a directed  $E$ -cycle of length  $m$ , say,  $b = b_0 \xrightarrow{E} b_1 \cdots \xrightarrow{E} b_m = b$ . The mapping  $h' : I_{f, f} \rightarrow J'$  can be constructed simply by extending  $h$  and mapping  $x_{e+i}$  to  $b_i$  for  $1 \leq i \leq m$ . ◀

► **Proposition 4.7.** *The monadic Datalog program given by the single rule  $\mathbf{Ans}(x) :- E(x, x)$  does not admit a generalized right-adjoint.*

*Proof.* Let  $P$  be the Boolean Datalog program in question. Note that  $\text{Unfoldings}(P, \mathbf{Ans})$  consists of a pointed structure that is c-acyclic but not acyclic. It does not admit a generalized right-adjoint: let  $J$  be the empty  $\mathbf{S}_{out}^P$ -instance, and suppose for the sake of contradiction that there is a finite set  $\{J_1, \dots, J_n\}$  such that, for all  $\mathbf{S}_{in}^P$ -instances  $I$ ,  $P(I) \rightarrow J$  iff  $I \rightarrow J_i$  for some  $i \leq n$ . Let  $I_c$  be the instance consisting of a single reflexive  $\mathbf{Ans}$ -edge of the form  $\mathbf{Ans}(a, a)$ . Clearly,  $P(I_c) \not\rightarrow J$ , and therefore,  $I_c \not\rightarrow J_i$ . That is,  $J_1, \dots, J_n$  do not contain a reflexive  $\mathbf{Ans}$ -edge. Next, let  $I_n$  be the instance that is an (irreflexive)  $\mathbf{Ans}$ -clique of size  $n$ , where  $n$  is any number greater than the size of each  $J_i$ . Then,  $I_n \not\rightarrow J_i$  (because if there was a homomorphism,  $J_i$  would contain a reflexive  $\mathbf{Ans}$ -edge), but, trivially,  $P(I_n) \rightarrow J$ . ◀

In the special case of Boolean non-recursive programs, there is a converse to Thm. 4.5:

► **Theorem 4.8.** *A Boolean non-recursive Datalog program has a generalized right-adjoint iff it is equivalent to a TAM Datalog program.*



This follows from results that we will prove in Section 5 (specifically, Thm. 5.4 together with Thm. 5.3). It also follows from a known result about Pultr functors, namely [17, Theorem 2.5], since Boolean non-recursive Datalog programs define Pultr functors.

► **Remark 4.9.** Thm. 4.8 does not hold for *recursive* Boolean Datalog programs. Indeed, consider the Boolean Datalog program with input schema  $\{R\}$  consisting of the rules

$$\begin{aligned} \text{Ans}() & \quad :- \quad \text{OddLengthPath}(x, x) \\ \text{OddLengthPath}(x, y) & \quad :- \quad R(x, y) \\ \text{OddLengthPath}(x, y) & \quad :- \quad R(x, z), R(z, u), \text{OddLengthPath}(u, y) \end{aligned}$$

It follows from well-known results in the CSP literature, (combined with Thm. 5.7 below), that  $P$  admits a generalized right-adjoint and that  $P$  is not equivalent to a monadic Datalog program. It follows by Lem. 3.8 that  $P$  is not equivalent to a TAM Datalog program.

► **Remark 4.10.** We note that the right adjoint  $\Omega_P(\cdot)$  of a TAM Datalog program  $P$  is in general not definable by a Datalog program. This follows trivially from the fact that  $\Omega_P(J)$  might consist of more than one structure if  $P$  is non connected and, in addition, It is not definable by a Datalog program even when  $P$  is connected as, in general, the domains of  $J$  and  $\Omega_P(J)$  do not need to be related. For instance, Example 4.4 contains an example where the domain of  $\Omega_P(J)$  is necessarily larger than that of  $J$ .

► **Remark 4.11.** Our definition of right-adjoints treats Datalog programs as functors in a flat category, while Lemma 2.3 naturally allows us to view Datalog programs as functors even in the non-flat category of instances and homomorphisms. This naturally raises a question of which functors between the “non-flat” categories allow right adjoints. This question has been answered by Pultr [22] up to some technical details, who described pairs of adjoint functors between categories of relational structures. Using either Pultr’s description, or by simple categorical methods, it can be seen that a Datalog program will rarely allow a proper right adjoint.

► **Remark 4.12.** While we are specifically interested in right-adjoints in this paper, one may also wonder what it means for a Datalog program to admit a (generalized) left-adjoint. Generalized left-adjoints for Datalog programs are closely related to query rewritings, as studied in the literature on data integration and data exchange. A Datalog program  $P$  has a generalized left-adjoint iff  $P$  is equivalent to a non-recursive Datalog program. Indeed, if  $P$  has a generalized left-adjoint  $\Theta$ , then, for each  $R \in \mathbf{S}_{out}^P$ , the  $\mathbf{S}_{in}^P$ -instances in  $\Theta(\{R(a_1, \dots, a_n)\})$  correspond to the members of  $\text{Unfoldings}(P, R)$  (cf. [17, 13]).

## 5 Right Adjoints and Homomorphism Dualities

For any set of instances  $X$ , let  $X^\uparrow = \{A \mid B \rightarrow A \text{ for some } B \in X\}$ , and let  $X^\downarrow = \{A \mid A \rightarrow B \text{ for some } B \in X\}$ . A *homomorphism duality* is a pair of sets of instances  $(F, D)$ , such that  $F^\uparrow$  is the complement of  $D^\downarrow$ . The same definition extends naturally to pointed instances. By a *finite homomorphism duality*, we mean a homomorphism duality  $(F, D)$  where  $F$  and  $D$  are finite sets. By a *tree duality*, we mean a homomorphism duality  $(F, D)$  where  $F$  is a (possibly infinite) set of (not-necessarily-connected) acyclic instances, and  $D$  is finite.

The study of such dualities originated in combinatorics (see [20]) motivated by its links to the structure of the homomorphism partial order, and the complexity of deciding the existence of homomorphism between graphs and, more generally, relational structures (a.k.a. constraint satisfaction problems or CSPs). Indeed, dualities have played an important role in the study of CSPs. In particular, it was shown [3] that the CSPs definable in FO are precisely those



whose template is the right-hand side of a finite duality. In a similar vein, the CSPs solvable by the well-known *arc-consistency* algorithm are precisely those whose template is the right-hand side of a tree duality. More generally, the CSPs solvable by local consistency methods are those whose template is the right-hand side of a homomorphism duality whose left-hand side consists of instances of bounded treewidth. See [8] for a survey on the connections between duality and consistency algorithms. In database theory, homomorphism dualities are used in the study of the unique characterizability and exact learnability of schema mappings and database queries [2, 9], closed-world rewritings of open-world queries [6], and extremal fitting algorithms [10].

► **Example 5.1.** Let  $\mathbf{S} = \{R\}$ , where  $R$  is a binary relation symbol, and let  $n \geq 1$ . Let  $L_n$  be the finite linear order of length  $n$ , and let  $P_{n+1}$  be the directed path of length  $n + 1$ . Then  $(\{P_{n+1}\}, \{L_n\})$  is a finite homomorphism duality.

► **Example 5.2.** Let  $\mathbf{S} = \{P_0, P_1, E\}$ , where  $P_0$  and  $P_1$  are unary and  $E$  is binary, and consider the two-element  $\mathbf{S}$ -instance  $I = \{P_0(0), P_1(1), E(0,0), E(1,1)\}$  (without distinguished elements). For all  $\mathbf{S}$ -instances  $J$ ,  $J \rightarrow I$  holds if and only if no connected component of  $J$  contains both a  $P_0$ -fact and a  $P_1$ -fact. This can be expressed in the form of a tree duality: let  $F$  be the set of all (acyclic) instances consisting of an oriented path that connects a  $P_0$ -node to a  $P_1$ -node (where, by an *oriented path* we mean a path  $a_1, a_2, \dots, a_n$  where, for each  $1 \leq i < n$ , either  $E(a_i, a_{i+1})$  or  $E(a_{i+1}, a_i)$ ). Then  $(F, \{I\})$  is a homomorphism duality.

► **Theorem 5.3** ([16, 9]). *Fix a schema  $\mathbf{S}$  and  $k \geq 0$ . Let  $F$  be any finite set of pairwise homomorphically incomparable  $k$ -ary pointed instances over  $\mathbf{S}$ . The following are equivalent:*

1. *There is a finite set of  $k$ -ary pointed instances  $D$  over  $\mathbf{S}$  such that  $(F, D)$  is a homomorphism duality.*
2. *Each pointed instance in  $F$  is homomorphically equivalent to a  $c$ -acyclic pointed instance. Moreover (for fixed  $\mathbf{S}$  and  $k$ ), given a set  $F$  of  $c$ -acyclic pointed instances, such a set  $D$  can be computed in *ExpTime*.<sup>3</sup>*

The following theorem establishes a close relationship between generalized right-adjoints and homomorphism dualities:<sup>4</sup>

► **Theorem 5.4.** *Let  $P$  be any Datalog program that has a generalized right-adjoint. Then, for each  $R \in \mathbf{S}_{out}^P$ , there is a finite set of pointed  $\mathbf{S}_{in}$ -instances  $D$  such that  $(\text{Unfoldings}(P, R), D)$  is a homomorphism duality.*

**Proof.** We may assume without loss of generality that  $\mathbf{S}_{out}^P = \{R\}$ . Let  $J$  be the  $\mathbf{S}_{out}^P$ -instance with  $\text{adom}(J) = \{b_1, \dots, b_k, c\}$  (for  $k = \text{arity}(R)$ ) containing all  $R$ -facts over  $\text{adom}(J)$  except  $R(b_1, \dots, b_k)$ . Let  $D = \{(J', \iota) \mid (J', \iota) \in \Omega_P(J), \mathbf{b}' \in \text{adom}(J')^k, \iota(\mathbf{b}') = \mathbf{b}\}$ , where  $\mathbf{b} = b_1, \dots, b_k$ . We claim that  $(\text{Unfoldings}(P, R), D)$  is a homomorphism duality. Let  $(C, \mathbf{c})$  be any  $\mathbf{S}_{in}$ -instance with  $k$  distinguished elements. Then an instance in  $\text{Unfoldings}(P, R)$  homomorphically maps to  $(C, \mathbf{c})$  iff  $R(\mathbf{c}) \in P(C)$  iff  $(P(C), \mathbf{c}) \not\rightarrow (J, \mathbf{b})$  iff (by the adjoint property)  $(C, \mathbf{c}) \not\rightarrow (J', \mathbf{b}')$  for all  $(J', \iota) \in \Omega_P(J)$  and  $\mathbf{b}'$  with  $\iota(\mathbf{b}') = \mathbf{b}$ . ◀

Thm. 5.4 shows that generalized right-adjoints can be used to construct duals. This approach was first used in [17] and [13], where right-adjoints of Pultr functors are applied to derive the dual of a tree. Thm. 5.4 can be viewed as extending results in [13] to a more general class of functors defined by recursive Datalog programs.

<sup>3</sup> The *ExpTime* bound is not explicitly stated in [9] but follows from results in that paper.

<sup>4</sup> Recall that  $\text{Unfoldings}(P, R)$  can be viewed as an infinite union of (canonical instances of) conjunctive queries that defines the output relation  $R$  (cf. Lemma 2.5).

For TAM Datalog programs  $P$ , the proof of Thm. 5.4 yields a ExpTime algorithm to compute  $D$  from  $(P, R)$  provided the arity of the relations in  $P$  is bounded. Since  $\text{Unfoldings}(P, R)$  consists of acyclic instances whenever  $P$  is a TAM Datalog program, this gives us a systematic way of constructing tree-dualities. In fact, every tree-duality can be obtained in this way:

► **Corollary 5.5.** *Let  $F$  be any set of acyclic pointed instances. The following are equivalent:*

1. *There is a finite set of pointed instances  $D$  such that  $(F, D)$  is a homomorphism duality*
2.  *$F^\uparrow = \text{Unfoldings}(P, R)^\uparrow$  for some TAM Datalog program  $P$  and  $R \in \mathbf{S}_{out}^P$ .*

**Proof.** From 1 to 2: It is well known that, for any finite set of pointed instances  $D$ , there is an MSO formula  $\phi$  that defines  $D^\downarrow$ . Hence, by duality,  $\neg\phi$  defines  $F^\uparrow$ . Furthermore, the fact that  $F$  consists of acyclic pointed instances implies that  $\neg\phi$  is tree-determined. Therefore, the direction 1 to 2 follows from Thm. 3.13. The direction from 2 to 1 follows from Thm. 5.4. ◀

It is possible to strengthen Corollary 5.5 by showing that the above conditions (1) and (2) are, in turn, equivalent to the fact that  $F^\uparrow = G^\uparrow$  for some regular set  $G$  of acyclic queries (where “regular” needs to be defined in a suitable way, as in [15]). This follows from the fact that Thm. 3.13 uses tree-automata as an intermediate step in the proof. We note that the special case of this equivalence for Boolean CQs over digraphs was proven in [15].

Thm. 5.4 also implies that every finite set of acyclic pointed instances  $F$  is the left-hand side of a finite homomorphism duality: it suffices to let  $P$  be the TAM Datalog program containing a single non-recursive rule for each  $(I, \mathbf{a}) \in F$ , whose canonical instance is  $(I, \mathbf{a})$ . Then, the unfoldings of  $P$  are, up to isomorphism, precisely the pointed instances in  $F$ . It follows from Thm. 5.4 that there is a finite set  $D$  such that  $(F, D)$  is a homomorphism duality. This provides an alternative proof of the characterization of left-hand sides of finite dualities given in [16] (i.e., the special case of Thm. 5.3 for structures without distinguished elements).

► **Remark 5.6.** In light of Thm. 5.3, it is natural to ask whether the above “dualities through adjoints” technique can be used to construct a finite homomorphism duality for any  $c$ -acyclic pointed instance. Prop. 4.7 shows that this is not possible. Note that the canonical instance of the rule of the program in Prop. 4.7 is  $(\{E(a, a)\}, a)$ , which is  $c$ -acyclic (but not acyclic).

For Boolean Datalog programs, the relationship between adjoints and dualities is tighter:

► **Theorem 5.7.** *For Boolean Datalog programs  $P$ , the following are equivalent:*

1.  *$P$  admits a generalized right-adjoint,*
2. *There is a finite set of pointed  $\mathbf{S}_{in}^P$ -instances  $D$  such that  $(\text{Unfoldings}(P, \text{Ans}), D)$  is a homomorphism duality.*

**Proof.** For every  $\mathbf{S}_{in}^P$ -instance  $I$ ,  $P(I)$  is either the empty instance, which we may denote as  $J_0$  or the instance consisting of the zero-ary fact  $\text{Ans}()$ , which we may denote as  $J_1$ . Let  $\Omega_P(J_0)$  be the set of all pairs  $(J', \iota)$  with  $J' \in D$  and  $\iota$  the empty partial function; and let  $\Omega_P(J_1) = \{(J', \iota)\}$  where  $J'$  is a single-element fully-connected  $\mathbf{S}_{in}^P$ -instance and  $\iota$  is the empty partial function. It is easy to see that  $\Omega_P$  is then a generalized right-adjoint for  $P$ . ◀

## 6 An Application: Generating Data Examples for Database Queries

We will now show-case one application of our results, which is concerned with the problem of generating data examples for database queries. A *data example* for a database query, informally, consists of a database instance  $I$  together with information about the output of the query when evaluated on  $I$ . Data examples can be a helpful tool in query debugging, query refinement, interactive query specification, and query learning. In each of these settings,

the question naturally comes up as to whether, for a given database query  $q$ , there exists a finite collection of data examples, such that, modulo logical equivalence,  $q$  is the *only* query (within some given class of queries) that fits the data examples. When this happens, we say that the collection of data examples in question *uniquely characterizes* the query  $q$ .

It was shown in [16, 9] that every “c-acyclic” union of conjunctive queries (UCQ) is indeed uniquely characterized by a finite collection of data examples. In fact, a UCQ is uniquely characterizable by a finite collection of data examples, if and only if it is equivalent to a c-acyclic UCQ. While this gives a precise answer to the question of unique characterizability, it can be cumbersome to use in practice. One of the reasons for this is that the data examples in question tend to look unnatural to a user. In particular, the existing algorithms for constructing data examples do not take integrity constraints into consideration. We will show here that the aforementioned results from [16, 9] can be adapted to the setting with integrity constraints, in such a way that all generated data examples satisfy the integrity constraints, provided that the integrity constraints are from a suitable, well-behaved class.

We will do this in three steps. First, we propose a suitable class of integrity constraints. Second, we study the existence of homomorphism dualities relative to a set of integrity constraints. Finally, we use this to construct uniquely characterizing examples for c-acyclic UCQs.

## 6.1 Tame sets of full TGDs

One of the most important classes of integrity constraints in databases is the class of *tuple-generating dependencies (TGDs)*. The results we will present will be concerned with a subclass of TGDs called *full TGDs*. A full TGD is a TGD without existential quantifiers. More precisely, a full TGD is a first-order sentence of the form  $\forall \mathbf{x}(\phi(\mathbf{x}) \rightarrow \psi(\mathbf{x}))$ , where  $\phi(\mathbf{x})$  and  $\psi(\mathbf{x})$  are conjunctions of relational atomic formulas, and each variable in  $\mathbf{x}$  occurs in  $\phi$ .

Every finite set of full TGDs naturally gives rise to a Datalog program. More precisely, for any set  $\Sigma$  of full TGDs over a schema  $\mathbf{S}$ , we will denote by  $P_\Sigma$  the Datalog program with  $\mathbf{S}_{in}^P = \{R_{in} \mid R \in \mathbf{S}\}$ ,  $\mathbf{S}_{out}^P = \{R_{out} \mid R \in \mathbf{S}\}$ , and  $\mathbf{S}_{aux}^P = \mathbf{S}$ , consisting of the full TGDs in  $\Sigma$  as Datalog rules (where  $\forall \mathbf{x}(\phi(\mathbf{x}) \rightarrow \psi(\mathbf{x}))$  becomes  $\psi(\mathbf{x}) :- \phi(\mathbf{x})$ ), plus the “copy constraints”  $R(\mathbf{x}) :- R_{in}(\mathbf{x})$  and  $R_{out}(\mathbf{x}) :- R(\mathbf{x})$  for each  $R \in \mathbf{S}$ .

Although the input and output schemas of  $P_\Sigma$  are renamings of  $\mathbf{S}$ , we will write  $P_\Sigma(I)$  even when  $I$  is an  $\mathbf{S}$ -instance instead of an  $\mathbf{S}_{in}$ , with the understanding that relation symbols are renamed in the obvious way; and similarly, we will freely treat the  $\mathbf{S}_{out}$ -instance  $P_\Sigma(I)$  as an  $\mathbf{S}$ -instance. The Datalog program  $P_\Sigma$  then “captures”  $\Sigma$  in the following sense:

► **Lemma 6.1.** *Let  $\Sigma$  be any finite set of full TGDs. For all instances  $I$ ,  $P_\Sigma(I)$  is the unique minimal instance  $J$  with  $I \subseteq J$  such that  $J \models \Sigma$ . In particular,  $P_\Sigma(I) = I$  when  $I \models \Sigma$ .*

We say that a finite set  $\Sigma$  of full TGDs is *tame* (or, TAM-equivalent) if  $P_\Sigma$  is equivalent to a TAM Datalog program. A full TGD is *monadic (tree-shaped)* if the associated Datalog program  $P_\Sigma$  (where  $\Sigma$  consists of the full TGD in question) is monadic (resp. tree shaped).

► **Example 6.2.** The following sets of full TGDs are tame:

- $\Sigma_1 = \{\forall xyz(R(x, y) \wedge R(y, z) \rightarrow R(x, z))\}$ . To see that  $P_{\Sigma_1}$  has a generalized right-adjoint, observe that it consists of the rules depicted on the left:

$$\begin{array}{ll}
 R(x, y) & :- R_{in}(x, y) \\
 R(x, z) & :- R(x, y), R(y, z) \\
 R_{out}(x, y) & :- R(x, y)
 \end{array}
 \qquad
 \begin{array}{ll}
 R(x, y) & :- R_{in}(x, y) \\
 R(x, z) & :- R(x, y), R_{in}(y, z) \\
 R_{out}(x, y) & :- R(x, y)
 \end{array}$$

$P_{\Sigma_1}$  is not a TAM Datalog program. However, it is equivalent to the program  $P'$  consisting of the rules depicted on the right. Note how we have replaced one occurrence of  $R$  by  $R_{in}$ . The equivalence of  $P_{\Sigma_1}$  and  $P'$  is easy to show. Furthermore,  $P'$  is a TAM Datalog program (where the articulation position of  $R$  is the second position).

- $\Sigma_2 = \{\forall xyz u(R(x, y) \wedge R(y, z) \wedge R(z, u) \rightarrow R(x, u)), \forall xy(R(x, y) \rightarrow R(y, x))\}$ . Although  $P_{\Sigma_2}$  is not a TAM Datalog program, it can be rewritten as one, using the same strategy as for  $\Sigma_1$ . We will return to this example later, in Remark 6.6.
- Every finite set  $\Sigma$  of monadic tree-shaped TGDs. Indeed,  $P_{\Sigma}$  is a TAM Datalog program.

► **Remark 6.3.** The above example involves adhoc arguments. We leave it as an open problem to define a large syntactic class of (sets of) TGDs that have a generalized right-adjoint, which includes  $\Sigma_1$ . Thm. 4.5 with Thm. 3.13 does imply that, for finite sets of tree-shaped TGDs  $\Sigma$ , if  $P_{\Sigma}$  is MSO-definable then  $\Sigma$  has a generalized right-adjoint.

Our main result in this section, namely Thm. 6.4, will apply to tame sets of full TGDs. The general strategy we develop here, however, can be extended to a larger class of integrity constraints that includes inclusion dependencies, as we will discuss in the conclusion section. This is beyond the scope of the present paper as it requires considering  $\exists$ Datalog programs (i.e., Datalog programs where existential quantifiers are allowed in rule heads).

## 6.2 Homomorphism dualities within restricted categories

Let  $\mathcal{C}$  be a class of instances over some schema (e.g., the class of transitive digraphs). We say that a pair  $(F, D)$ , with  $F, D \subseteq \mathcal{C}$ , is a *homomorphism duality within  $\mathcal{C}$*  if  $F^\uparrow \cap \mathcal{C}$  is the complement of  $D^\downarrow \cap \mathcal{C}$  relative to  $\mathcal{C}$ . In what follows we will also speak of homomorphism dualities *with respect to a theory  $\Sigma$* . By this, we mean homomorphism dualities w.r.t. the class of instances defined by  $\Sigma$ . The next result shows how to obtain finite homomorphism dualities within  $\mathcal{C}$ , for classes  $\mathcal{C}$  that are definable by a tame set of full TGDs.

► **Theorem 6.4.** *Let  $\Sigma$  be a tame set of full TGDs. Let  $F$  be any finite set of pointed instances. If each member of  $F$  is of the form  $(P_{\Sigma}(A), \mathbf{a})$  for some  $c$ -acyclic pointed instance  $(A, \mathbf{a})$ , then  $F$  is the left-hand side of a finite duality w.r.t.  $\Sigma$ .*

**Proof.** Let  $F'$  be the finite set of  $c$ -acyclic pointed instances such that  $F = \{(P_{\Sigma}(I), \mathbf{a}) \mid (I, \mathbf{a}) \in F'\}$ . Let  $D$  be the finite set such that  $(F', D)$  is a homomorphism duality, given by Thm. 5.3. Let  $D' = \{(P_{\Sigma}(B'), \mathbf{b}') \mid (B, \mathbf{b}) \in D, (B', \iota) \in \Omega_{P_{\Sigma}}(B), \iota(\mathbf{b}') = \mathbf{b}\}$ . Note that  $D'$  consists of pointed instances satisfying  $\Sigma$ . We will show that  $(F, D')$  is a homomorphism duality w.r.t.  $\Sigma$ . Let  $(C, \mathbf{c})$  be a pointed instance with  $C \models \Sigma$ . Then:

$$(C, \mathbf{c}) \in F^\uparrow \Leftrightarrow (C, \mathbf{c}) \in F'^\uparrow \Leftrightarrow (C, \mathbf{c}) \notin D^\downarrow \Leftrightarrow (C, \mathbf{c}) \notin D'^\downarrow$$

The first equivalence holds by Lem. 2.3 and the fact that  $P_{\Sigma}(C) = C$ . The second equivalence holds by the duality assumption. It remains to prove the third equivalence.

From left to right: By contraposition. Suppose that  $(C, \mathbf{c}) \rightarrow (P_{\Sigma}(B'), \mathbf{b}')$  for some  $(B, \mathbf{b}) \in D, (B', \iota) \in \Omega_{P_{\Sigma}}(B)$ , and  $\iota(\mathbf{b}') = \mathbf{b}$ . Trivially, we have  $id : (B', \mathbf{b}') \rightarrow (B', \mathbf{b}')$ . It follows by the generalized adjoint property that  $(P_{\Sigma}(B'), \mathbf{b}') \rightarrow (B, \iota(\mathbf{b}'))$ . Therefore, by transitivity, and since  $\iota(\mathbf{b}') = \mathbf{b}$ , we have  $(C, \mathbf{c}) \rightarrow (B, \mathbf{b})$  and therefore  $(C, \mathbf{c}) \in D^\downarrow$ .

From right to left: Again, by contraposition. Assume  $(C, \mathbf{c}) \in D'^\downarrow$ . Since  $P_{\Sigma}(C) = C$ , it follows that  $(P_{\Sigma}(C), \mathbf{c}) \rightarrow (B, \mathbf{b})$  for some  $(B, \mathbf{b}) \in D$ . It follows by the adjoint property that  $(C, \mathbf{c}) \rightarrow (B', \mathbf{b}')$  for some  $(B, \mathbf{b}) \in D, (B', \iota) \in \Omega_{P_{\Sigma}}(B)$ , and  $\mathbf{b}' \in \iota^{-1}(\mathbf{b})$ . Then also  $(C, \mathbf{c}) \rightarrow (P_{\Sigma}(B'), \mathbf{b}')$ . This means that  $(C, \mathbf{c}) \in F'^\uparrow$ . ◀

Regarding complexity, consider the case where  $\Sigma$  is a fixed tame set of full TGDs (not treated as part of the input). Then the proof of Thm. 6.4 yields a 2ExpTime algorithm for computing the dual set  $D$  from  $F$ , assuming  $F$  is specified by the underlying set of  $c$ -acyclic instances  $(A, \mathbf{a})$ . Thus, for instance, for the class of transitive digraphs (which, as we saw earlier, is captured by a TAM Datalog program), we have a 2ExpTime-algorithm for constructing duals for digraphs that are specified as the transitive closure of an acyclic digraph.

The only prior results regarding homomorphism dualities for restricted classes of structures that we are aware of, are for undirected graphs and for finite algebras. An undirected graph can be viewed as an instance over a schema  $\mathbf{S}$  consisting of a single binary relation symbol  $E$ , satisfying the integrity constraints  $\forall xy(E(x, y) \rightarrow E(y, x))$  and  $\forall x \neg E(x, x)$ . It is known that the category of undirected graphs and homomorphisms has no finite dualities, up to homomorphic equivalence, other than the trivial duality  $(\{K_2\}, \{K_1\})$ , where  $K_1$  and  $K_2$  are the 2-element clique and the empty graph, respectively (cf. [20]). Similarly, a finite algebra of a similarity type  $\sigma$  can be viewed as an  $\mathbf{S}$ -instance, with  $\mathbf{S} = \{R_f \mid f \in \sigma\}$  satisfying  $\Sigma = \{\forall \mathbf{x} \exists y R_f(\mathbf{x}, y), \forall \mathbf{x} y z (R_f(\mathbf{x}, y) \wedge R_f(\mathbf{x}, z) \rightarrow y = z) \mid f \in \sigma\}$ , and, again, it is known that, in the category of finite algebras, no non-trivial finite dualities exist [4]. Note that both in the case of undirected graphs (viewed as symmetric and irreflexive relational structures) and in the case of finite algebras, all non-trivial structures in question are cyclic.

For the special case of monadic tree-shaped TGDs, we can prove a converse to Thm. 6.4:

► **Theorem 6.5.** *Let  $\Sigma$  be any set of monadic tree-shaped TGDs. Let  $F$  be any finite set of pairwise homomorphically-incomparable pointed instances  $(A, \mathbf{a})$  with  $A \models \Sigma$ . Then, the following are equivalent:*

1.  $F$  is the left hand side of a finite duality w.r.t.  $\Sigma$ ,
2. Each  $(A, \mathbf{a}) \in F$  is homomorphically equivalent to  $(P_\Sigma(A'), \mathbf{a})$  for some  $c$ -acyclic  $(A', \mathbf{a})$ .

► **Remark 6.6.** Thm. 6.5 cannot be lifted to arbitrary tame sets of full TGDs. Consider again the tame set of full TGDs  $\Sigma = \{\forall xyz (R(x, y) \wedge R(y, z) \wedge R(z, x) \rightarrow R(x, x)), \forall xy (R(x, y) \rightarrow R(y, x))\}$  from Example 6.2. Let  $A$  be the instance (without distinguished elements)  $\{R(a, a)\}$ , and let  $B$  be the instance  $\{R(a, b), R(b, a)\}$ . Then  $(\{A\}, \{B\})$  is a homomorphism duality w.r.t.  $\Sigma$ . Indeed, let  $C$  be an instance satisfying  $\Sigma$  and assume that  $A \not\rightarrow C$  (i.e,  $C$  has no loop). Since  $C$  satisfies  $\Sigma$  it follows that  $C$  has no odd cycle and, hence, is homomorphic to  $B$ . However, it is easy to see that every instance  $A'$  satisfying  $P_\Sigma(A') = A$  must have a cycle.

### 6.3 Uniquely Characterizing Examples for Database Queries

By a *collection of labeled examples* for a  $k$ -ary query, we mean a pair  $(E^+, E^-)$  of finite sets of pointed instances with  $k$  distinguished elements.<sup>5</sup> A UCQ  $q$  fits such  $(E^+, E^-)$  if  $\mathbf{a} \in q(A)$  for all  $(A, \mathbf{a}) \in E^+$ , and  $\mathbf{a} \notin q(A)$  for all  $(A, \mathbf{a}) \in E^-$ . We say that two UCQs  $q, q'$  (over the same schema  $\mathbf{S}$ ) are *equivalent w.r.t.  $\Sigma$* , where  $\Sigma$  is a first-order theory, if for all  $I \in \text{Inst}[\mathbf{S}]$  with  $I \models \Sigma$ ,  $q(I) = q'(I)$ . A collection of labeled examples  $(E^+, E^-)$  *uniquely characterizes* a UCQ  $q$  w.r.t.  $\Sigma$ , if  $q$  fits  $(E^+, E^-)$ , and every UCQ that fits  $(E^+, E^-)$  is equivalent to  $q$  w.r.t.  $\Sigma$ .

<sup>5</sup> Data examples can also be defined as pairs  $(I, q(I))$  of an input instance and the complete query output. This would not change our results. Note that such a data example  $(I, q(I))$  can be equivalently represented by all positive examples  $(I, \mathbf{a})$  for  $\mathbf{a} \in q(I)$  and negative examples  $(I, \mathbf{a})$  for  $\mathbf{a} \in \text{dom}(I)^k \setminus q(I)$ .

► **Lemma 6.7.** *Let  $\mathbf{S}$  be any schema and  $\Sigma$  a FO theory over  $\mathbf{S}$ . Let  $q$  be a UCQ over  $\mathbf{S}$ , and let  $E^+, E^-$  be finite sets of pointed instances  $(I, \mathbf{a})$  with  $I \models \Sigma$ . The following are equivalent:*

1. *The collection of labeled examples  $(E^+, E^-)$  uniquely characterizes  $q$  w.r.t.  $\Sigma$*
2.  *$q$  fits  $(E^+, E^-)$  and  $(E^+, E^-)$  is a finite homomorphism duality w.r.t.  $\Sigma$ .*

► **Theorem 6.8.** *Let  $\Sigma$  a tame set of full TGDs over a schema  $\mathbf{S}$ . Every c-acyclic UCQ  $q$  over  $\mathbf{S}$  is uniquely characterized w.r.t.  $\Sigma$  by a collection of labeled examples satisfying  $\Sigma$ .*

**Proof.** Let  $E^+$  be the set of pointed instances  $(P_\Sigma(I), \mathbf{a})$ , for  $(I, \mathbf{a})$  a (c-acyclic) canonical instance of a CQ in  $q$ . By Thm. 6.4, there is a finite set  $E^-$  such that  $(E^+, E^-)$  is a homomorphism duality w.r.t.  $\Sigma$ . By Lem. 6.7,  $(E^+, E^-)$  uniquely characterizes  $q$  w.r.t.  $\Sigma$ . ◀

The proof of Thm. 6.8 yields a 2ExpTime upper bound for computing uniquely characterizing examples for a given c-acyclic UCQ, relative to a fixed tame set of full TGDs. It is not known whether this is optimal. In the absence of integrity constraints, uniquely characterizing examples for a c-acyclic UCQ can be constructed in ExpTime and this is known to be optimal since they are in general exponential in size. In the case of c-acyclic CQs, uniquely characterizing examples can be constructed in polynomial time [9].

## 7 Conclusion

We introduced a new fragment of Datalog, TAM Datalog, that is semantically well-behaved (closed under composition and having a natural semantic characterization) and admits generalized right-adjoints. We used this result to obtain a method for constructing uniquely characterizing data examples for c-acyclic UCQs in the presence of integrity constraints (where the data examples are required to satisfy the integrity constraints). Generalized right-adjoints for Datalog programs seem potentially useful in other contexts as well, such as in tasks involving reasoning about a hidden database instance based on an exposed view (cf. [5]).

In a companion paper (cf. [11]), we further extend our study to  $\exists$ Datalog (the extension of Datalog where existential quantifiers are allowed in rule heads), and we show that linear  $\exists$ Datalog programs have right-adjoints. This is then used to extend our results on uniquely characterizing data examples to the case with (a weakly acyclic set of) inclusion dependencies.

We leave as open problems for future research: (i) obtaining tight complexity bounds for the task of constructing uniquely characterizing data examples for CQs and UCQs in the presence of a tame set of full TGDs; (ii) identifying better syntactic criteria that guarantees that a given finite set of full TGDs is tame; (iii) extending our results to the case with functional dependencies<sup>6</sup>; and (iv) studying which logic (or, more generally, formalism) is necessary to define the right adjoint  $\Sigma_P(\cdot)$  (see also Remark 4.10).

---

## References

- 1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison Wesley, 1995.
- 2 Bogdan Alexe, Balder ten Cate, Phokion G. Kolaitis, and Wang-Chiew Tan. Characterizing schema mappings via data examples. *ACM Trans. Database Syst.*, 36(4):23:1–23:48, dec 2011. doi:10.1145/2043652.2043656.

---

<sup>6</sup> Recent results in [18] show that  $\mathcal{ELI}$ -concepts (equivalently: unary connected acyclic CQs over a schema with only binary relations) are uniquely characterizable in the presence of functional dependencies, and that uniquely characterizing examples can be computed for them in polynomial time.



- 3 Albert Atserias. On digraph coloring problems and treewidth duality. *Eur. J. Comb.*, 29(4):796–820, 2008. doi:10.1016/j.ejc.2007.11.004.
- 4 Richard N. Ball, Jaroslav Nešetřil, and Aleš Pultr. Dualities in full homomorphisms. *European Journal of Combinatorics*, 31(1):106–119, 2010. doi:10.1016/j.ejc.2009.04.004.
- 5 Michael Benedikt, Pierre Bourhis, Balder Ten Cate, Gabriele Puppis, and Michael Vanden Boom. Inference from visible information and background knowledge. *ACM Trans. Comput. Logic*, 22(2), jun 2021. doi:10.1145/3452919.
- 6 Meghyn Bienvenu, Balder Ten Cate, Carsten Lutz, and Frank Wolter. Ontology-based data access: A study through Disjunctive Datalog, CSP, and MMSNP. *ACM Trans. Database Syst.*, 39(4), dec 2015. doi:10.1145/2661643.
- 7 Manuel Bodirsky, Simon Knäuer, and Sebastian Rudolph. Datalog-expressibility for monadic and guarded second-order logic. In *ICALP 2021*, pages 120:1–120:17, 2021. doi:10.4230/LIPIcs.ICALP.2021.120.
- 8 Andrei A. Bulatov, Andrei A. Krokhin, and Benoît Larose. Dualities for constraint satisfaction problems. In Nadia Creignou, Phokion G. Kolaitis, and Heribert Vollmer, editors, *Complexity of Constraints - An Overview of Current Research Themes [Result of a Dagstuhl Seminar]*, volume 5250 of *LNCS*, pages 93–124. Springer, 2008. doi:10.1007/978-3-540-92800-3\_5.
- 9 Balder ten Cate and Victor Dalmau. Conjunctive queries: Unique characterizations and exact learnability. *ACM Trans. Database Syst.*, 47(4), nov 2022. doi:10.1145/3559756.
- 10 Balder ten Cate, Victor Dalmau, Maurice Funk, and Carsten Lutz. Extremal fitting problems for conjunctive queries. In *Proceedings of the 42nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS'23*, 2023. doi:10.1145/3584372.3588655.
- 11 Balder ten Cate, Víctor Dalmau, and Jakub Opršal. Right-adjoints for datalog programs, and homomorphism dualities over restricted classes, 2023. doi:10.48550/arXiv.2302.06366.
- 12 Surajit Chaudhuri and Moshe Y Vardi. On the equivalence of recursive and nonrecursive Datalog programs. *Journal of Computer and System Sciences*, 54(1):61–78, 1997. doi:10.1006/jcss.1997.1452.
- 13 Victor Dalmau, Andrei Krokhin, and Jakub Opršal. Functors on relational structures which admit both left and right adjoints, 2023. doi:10.48550/arXiv.2302.13657.
- 14 Víctor Dalmau and Jakub Opršal. Local consistency as a reduction between constraint satisfaction problems, 2023. doi:10.48550/arXiv.2301.05084.
- 15 Péter L. Erdős, Dömötör Pálvölgyi, Claude Tardif, and Gábor Tardos. Regular families of forests, antichains and duality pairs of relational structures. *Comb.*, 37(4):651–672, 2017. doi:10.1007/s00493-015-3003-4.
- 16 Jan Foniok, Jaroslav Nešetřil, and Claude Tardif. Generalised dualities and maximal finite antichains in the homomorphism order of relational structures. *Eur. J. Comb.*, 29(4):881–899, 2008. doi:10.1016/j.ejc.2007.11.017.
- 17 Jan Foniok and Claude Tardif. Digraph functors which admit both left and right adjoints. *Discrete Mathematics*, 338(4):527–535, 2015. doi:10.1016/j.disc.2014.10.018.
- 18 Maurice Funk, Jean Christoph Jung, and Carsten Lutz. Frontiers and exact learning of ELI queries under DL-Lite ontologies. In *Proceedings of the 31st International Joint Conference on Artificial Intelligence, IJCAI-22*, 2022. doi:10.24963/ijcai.2022/364.
- 19 Georg Gottlob and Christoph Koch. Monadic Datalog and the expressive power of languages for web information extraction. *J. ACM*, 51(1):74–113, jan 2004. doi:10.1145/962446.962450.
- 20 Pavol Hell and Jaroslav Nešetřil. *Graphs and homomorphisms*, volume 28 of *Oxford lecture series in mathematics and its applications*. Oxford University Press, 2004.
- 21 Andrei A. Krokhin, Jakub Opršal, Marcin Wrochna, and Stanislav Živný. Topology and adjunction in promise constraint satisfaction. *SIAM Journal on Computing*, 52(1):38–79, 2023. doi:10.1137/20M1378223.
- 22 Aleš Pultr. The right adjoints into the category of relational systems. In *Reports of the Midwest Category Seminar IV*, volume 137 of *Lecture Notes in Mathematics*, pages 100–113. Springer, 1970.
- 23 Sebastian Rudolph and Markus Krötzsch. Flag & check: Data access with monadically defined queries. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS '13*, pages 151–162. ACM, 2013. doi:10.1145/2463664.2465227.



# On the Convergence Rate of Linear Datalog<sup>◦</sup> over Stable Semirings

**Sungjin Im**

University of California, Merced, CA, USA

**Benjamin Moseley**

Carnegie Mellon University, Pittsburgh, PA, USA

**Hung Ngo**

RelationalAI, Berkeley, CA, USA

**Kirk Pruhs**

University of Pittsburgh, Pittsburgh, PA, USA

---

## Abstract

Datalog<sup>◦</sup> is an extension of Datalog, where instead of a program being a collection of union of conjunctive queries over the standard Boolean semiring, a program may now be a collection of sum-product queries over an arbitrary commutative partially ordered pre-semiring. Datalog<sup>◦</sup> is more powerful than Datalog in that its additional algebraic structure allows for supporting recursion with aggregation. At the same time, Datalog<sup>◦</sup> retains the syntactic and semantic simplicity of Datalog: Datalog<sup>◦</sup> has declarative least fixpoint semantics. The least fixpoint can be found via the naïve evaluation algorithm that repeatedly applies the immediate consequence operator until no further change is possible.

It was shown in [10] that, when the underlying semiring is  $p$ -stable, then the naïve evaluation of any Datalog<sup>◦</sup> program over the semiring converges in a finite number of steps. However, the upper bounds on the rate of convergence were exponential in the number  $n$  of ground IDB atoms.

This paper establishes polynomial upper bounds on the convergence rate of the naïve algorithm on **linear** Datalog<sup>◦</sup> programs, which is quite common in practice. In particular, the main result of this paper is that the convergence rate of linear Datalog<sup>◦</sup> programs under any  $p$ -stable semiring is  $O(pn^3)$ . Furthermore, we show a matching lower bound by constructing a  $p$ -stable semiring and a linear Datalog<sup>◦</sup> program that requires  $\Omega(pn^3)$  iterations for the naïve iteration algorithm to converge. Next, we study the convergence rate in terms of the number of elements in the semiring for linear Datalog<sup>◦</sup> programs. When  $L$  is the number of elements, the convergence rate is bounded by  $O(pn \log L)$ . This significantly improves the convergence rate for small  $L$ . We show a nearly matching lower bound as well.

**2012 ACM Subject Classification** Theory of computation → Database query languages (principles)

**Keywords and phrases** Datalog, convergence rate, semiring

**Digital Object Identifier** 10.4230/LIPIcs.ICDT.2024.11

**Funding** Moseley was supported in part by a Google Research Award, an Inform Research Award, a Carnegie Bosch Junior Faculty Chair, and NSF grants CCF-2121744 and CCF-1845146. Pruhs was supported in part by NSF grants CCF-1907673, CCF-2036077, CCF-2209654 and an IBM Faculty Award. Im was supported in part by NSF grants CCF-1844939 and CCF-2121745.

## 1 Introduction

In order to express common recursive computations with aggregates in modern data analytics while retaining the syntactic and semantic simplicity of Datalog, [10] introduced Datalog<sup>◦</sup>, an extension of Datalog that allows for aggregation and recursion over an arbitrary *commutative partially ordered pre-semiring* (POPS). Datalog is exactly Datalog<sup>◦</sup> over the Boolean semiring. Like Datalog, Datalog<sup>◦</sup> has a declarative least fixpoint semantics, and the least fixpoint can



© Sungjin Im, Benjamin Moseley, Hung Ngo, and Kirk Pruhs;  
licensed under Creative Commons License CC-BY 4.0

27th International Conference on Database Theory (ICDT 2024).

Editors: Graham Cormode and Michael Shekelyan; Article No. 11; pp. 11:1–11:20

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

be found via the naïve iteration algorithm that repeatedly applies the immediate consequence operator until no further change is possible. Moreover, its additional algebraic structure allows for common recursions with aggregations.

However unlike Datalog, the naïve evaluation of a Datalog<sup>◦</sup> program may not always converge in a finite number of steps. The convergence of a Datalog<sup>◦</sup> program over a given POPS can be studied through its “core semiring”, which is where we focus our attention on in this paper. This paper will only study Datalog<sup>◦</sup> programs over commutative semirings, referring the readers to [10] for the generality of POPS.

It is known that the commutative semirings for which the iterative evaluation of Datalog<sup>◦</sup> programs is guaranteed to converge are exactly those semirings that are stable [10]. A semiring is  $p$ -stable if the number of iterations required for any one-variable recursive linear Datalog<sup>◦</sup> program to reach a fixed point is at most  $p$ , and a semiring is stable if there exists a  $p$  for which it is  $p$ -stable. Further, every non-stable semiring has a simple (linear) Datalog<sup>◦</sup> program with one variable with the property that the iterative evaluation of this program over that semiring will not converge. Thus it is natural to concentrate on Datalog<sup>◦</sup> programs over stable semirings. Previously, the best known upper bound on the convergence rate, which is the number of iterations until convergence, is  $\sum_{i=1}^n (p+2)^i = \Theta(p^n)$  steps, where  $n$  is the number of ground atoms for the IDB’s that ever have a nonzero value at some point in the iterative evaluation of the Datalog<sup>◦</sup> program, and the underlying semiring is  $p$ -stable. In contrast there are no known lower bounds that show that iterative evaluation requires an exponential (in the parameter  $n$ ) number of steps to reach convergence.

The exact general upper bound on the convergence rate of Datalog<sup>◦</sup> programs over  $p$ -stable semirings is open, even for the special case of linear Datalog<sup>◦</sup> programs. Linear Datalog<sup>◦</sup> programs are quite common in practice.<sup>1</sup> Thus, in this paper we focus on this “easiest” case where the exact convergence rate is not known.

Currently, the best known upper bound on the convergence rate of linear Datalog<sup>◦</sup> programs over  $p$ -stable semirings is  $\sum_{i=1}^n (p+1)^i$  steps. This bound is unsatisfactory in the following sense. The prototypical example of a  $p$ -stable semiring is the tropical semiring  $\text{Trop}_p^+$ , (see Section 2 for a definition of  $\text{Trop}_p^+$ ); in this case, it is known that the naïve algorithm converges in  $O(pn)$  steps for linear Datalog<sup>◦</sup> programs [10]. These results leave open the possibility that the convergence rate of the naïve algorithm on linear Datalog<sup>◦</sup> programs over  $p$ -stable semirings could be exponentially smaller than the best known guarantee.

This paper seeks to obtain tighter bounds on the convergence rate of naïve evaluation of linear Datalog<sup>◦</sup> programs. As the iterative evaluation of Datalog<sup>◦</sup> programs is a reasonably natural and important algorithm/process, bounding the running time of this process is of both theoretical interests and practical interests. In practice, a known upper bound on the convergence rate allows the database system to determine *before hand* an upper bound on the number of iterations that will be required to evaluation a particular Datalog<sup>◦</sup> program.

## 1.1 Background

Before stating our main results, we need to back up to set the stage a bit. A (traditional) Datalog program  $\Pi$  consists of a set of rules of the form:

$$R_0(\mathbf{X}_0) \text{ :- } R_1(\mathbf{X}_1) \wedge \cdots \wedge R_m(\mathbf{X}_m) \tag{1}$$

<sup>1</sup> For example, we can express transitive closure, all-pairs-shortest-paths, or weakly connected components in linear Datalog<sup>◦</sup>.

where  $R_0, \dots, R_m$  are predicate names (not necessarily distinct) and each  $\mathbf{X}_i$  is a tuple of variables and/or constants. The atom  $R_0(\mathbf{X}_0)$  is called the head, and the conjunction  $R_1(\mathbf{X}_1) \wedge \dots \wedge R_m(\mathbf{X}_m)$  is called the body of the rule. Multiple rules with the same head are interpreted as a disjunction. A predicate that occurs in the head of some rule in  $\Pi$  is called an *intensional database predicate* or IDB, otherwise it is called an *extensional database predicate* or EDB. The EDBs form the input database, and the IDBs represent the output instance computed by  $\Pi$ . The finite set of all constants occurring in an EDB is called the *active domain*, and denoted  $\text{ADom}$ . A Datalog program is *linear* if every rule has at most one IDB predicate in the body.

There is an implicit existential quantifier over the body for all variables that appear in the body but not in the head, where the domain of the existential quantifier is  $\text{ADom}$ . In a linear Datalog program every conjunction has at most one IDB.

► **Example 1.** *The textbook example of a linear Datalog program is the following one, which computes the transitive closure of a directed graph, defined by the edge relation  $E(X, Y)$ :*

$$\begin{aligned} T(X, Y) &:- E(X, Y) \\ T(X, Y) &:- T(X, Z) \wedge E(Z, Y) \end{aligned}$$

Here  $E$  is an EDB predicate,  $T$  is an IDB predicate, and  $\text{ADom}$  consists of the vertices in the graph. The other way to write this program is to write it as a union of conjunctive queries (UCQs), where the quantifications are explicit:

$$T(X, Y) :- E(X, Y) \vee \exists_Z (T(X, Z) \wedge E(Z, Y)) \quad (2)$$

A Datalog program can be thought of as a function (called the *immediate consequence operator*, or *ICO*) that maps a set of ground IDB atoms to a set of ground IDB atoms. Every rule in the program is an inference rule that can be used to infer new ground IDB atoms from old ones. For a particular EDB instance, this function has a unique least fixpoint which can be obtained via repeatedly applying the ICO until a fixpoint is reached [1]. This least fixpoint is the semantics of the given Datalog program. The algorithm is called the *naïve evaluation algorithm*, which converges in a polynomial number of steps in the size of the input database, given that the program is of fixed size.

Like Datalog programs, a  $\text{Datalog}^\circ$  program consists of a set of rules, where the unions of conjunctive queries are now replaced with sum-product queries over a commutative semiring  $\mathcal{S} = (\mathcal{S}, \oplus, \otimes, 0, 1)$ ; see Section 2. Each rule has the form:

$$R_0(X_0) :- \bigoplus R_1(\mathbf{X}_1) \otimes \dots \otimes R_m(\mathbf{X}_m) \quad (3)$$

where sum ranges over the active  $\text{ADom}$  of the variables not in  $X_0$ . Further each ground atom in an EDB predicate or IDB predicate is associated with an element of the semiring  $\mathcal{S}$ , and the element associated with a tuple in an EDB predicate is specified in the input. The EDBs form the input database, and the ground atoms in the IDB's that have an associated semiring value that is nonzero represent the output instance computed by the  $\text{Datalog}^\circ$  program. Note that in a Datalog program the ground atom present in the input or output databases can be thought of as those that are associated with the element 1 in the standard Boolean semiring. A Datalog program is a  $\text{Datalog}^\circ$  program over the Boolean semiring  $\mathcal{S} = (\{\text{true}, \text{false}\}, \vee, \wedge, \text{false}, \text{true})$ . Again a  $\text{Datalog}^\circ$  program is linear if every rule has no more than one IDB predicate in its body.

## 11:4 On the Convergence Rate of Linear Datalog<sup>o</sup> over Stable Semirings

► **Example 2.** A simple example of a linear Datalog<sup>o</sup> program is the following,

$$T(X, Y) :- E(X, Y) \oplus \bigoplus_Z T(X, Z) \otimes E(Z, Y), \quad (4)$$

which is (2) with  $(\vee, \wedge, \exists_Z)$  replaced by  $(\oplus, \otimes, \bigoplus_Z)$ .

When interpreted over the Boolean semiring, we obtain the transitive closure program from Example 1. When interpreted over the tropical semiring  $\text{Trop}^+ = (\mathbb{R}_+ \cup \{\infty\}, \min, +, \infty, 0)$ , we have the All-Pairs-Shortest-Path (APSP) program, which computes the shortest path length  $T(X, Y)$  between all pairs  $X, Y$  of vertices in a directed graph specified by an edge relation  $E(X, Y)$ , where the semiring element associated with  $E(X, Y)$  is the length of the directed edge  $(X, Y)$  in the graph:

$$T(X, Y) :- \min \left( E(X, Y), \min_Z (T(X, Z) + E(Z, Y)) \right) \quad (5)$$

A Datalog<sup>o</sup> program can be thought of as an immediate consequence operator (ICO). To understand how the ICO works in Datalog<sup>o</sup>, consider a rule with head  $R$  and let  $A$  be a ground atom for  $R$  with associated semiring value  $y$ , and assume that for  $A$  the body of this rule evaluates to the semiring element  $x$ . As a result of this, the ICO associates  $A$  with  $x \oplus y$ . Note that the functioning of the Datalog<sup>o</sup> ICO, when the semiring is the standard Boolean semiring, is identical to how the Datalog ICO functions. The iterative evaluation of a Datalog<sup>o</sup> program works in rounds/steps, where initially the semiring element 0 is associated with each possible ground atom in an IDB, and on each round the ICO is applied to the current state. So in the context of the Datalog<sup>o</sup> program in Example 1, if  $(a, b)$  was a ground atom in  $T$  with associated semiring element  $x$ , meaning that the shortest known directed path from vertex  $a$  to vertex  $b$  has length  $x$ , and  $(b, c)$  was a ground atom in  $E$  with associated semiring element  $y$ , meaning that there is a directed edge from  $b$  to  $c$  with length  $y$ , then the ICO would make the semiring element associated with  $A$  the minimum (as minimum is the addition operation in the tropical semiring) of its current value and  $x + y$  (as normal additional is the multiplication operation in the tropical semiring).

Since the final associated semiring values of the ground atoms in an IDB are not initially known, it is natural to think of them as (IDB) variables. Then the grounded version of the ICO of a Datalog<sup>o</sup> program is a map  $\mathbf{f} : S^n \rightarrow S^n$ , where  $n$  is the number of ground atoms for the IDB's that ever have a nonzero value at some point in the iterative evaluation of the Datalog<sup>o</sup> program. For instance, in (5), there would be one variable for each pair  $(x, y)$  of vertices where there is a directed path from  $x$  to  $y$  in the graph. So the grounded version of the ICO of a Datalog<sup>o</sup> program has the following form:

$$\begin{aligned} X_1 &:- f_1(X_1, \dots, X_n) \\ &\dots \\ X_n &:- f_n(X_1, \dots, X_n) \end{aligned} \quad (6)$$

where the  $X_i$ 's are the IDB variables, and  $f_i$  is the component of  $\mathbf{f}$  corresponding to the IDB variable  $X_i$ . Note that each component function  $f_i$  is a multivariate polynomial in the IDB variables of degree at most the maximum number of terms in any product in the body of some rule in the Datalog<sup>o</sup> program. After  $q$  iterations of the iterative evaluation of a Datalog<sup>o</sup> program, the semiring value associated with the ground atom corresponding to  $X_i$  will be:

$$f_i^{(q)}(\mathbf{0}) \quad (7)$$

► **Definition 1** (*p*-stability). Given a semiring  $\mathcal{S} = (S, \oplus, \otimes, 0, 1)$  and  $u \in S$ , let  $u^{(p)} := 1 \oplus u \oplus u^2 \oplus \dots \oplus u^p$ , where  $u^i := u \otimes u \otimes \dots \otimes u$  ( $i$  times). Then  $u \in S$  is *p*-stable if  $u^{(p)} = u^{(p+1)}$ , and semiring  $\mathcal{S}$  is *p*-stable if every element  $u \in S$  is *p*-stable.

► **Definition 2** (Stability index and convergence rate). A function  $f : S^n \rightarrow S^n$  is *p*-stable if  $f^{(p+1)}(\mathbf{0}) = f^{(p)}(\mathbf{0})$ , where  $f^{(k)}$  is the  $k$ -fold composition of  $f$  with itself. The stability index of  $f$  is the smallest  $p$  such that  $f$  is *p*-stable. The convergence rate of a  $\text{Datalog}^\circ$  program is the stability index of its ICO.

The following bounds on the rate of convergence of a general (multivariate) polynomial function  $f : S^n \rightarrow S^n$ , where  $\mathcal{S}$  is *p*-stable; this result naturally infers bounds on the convergence rate of  $\text{Datalog}^\circ$  programs over *p*-stable semirings.

► **Theorem 3** ([10]). The convergence rate of a  $\text{Datalog}^\circ$  program over a *p*-stable commutative semiring is at most  $\sum_{i=1}^n (p+2)^i$ . Further, the convergence rate is at most  $\sum_{i=1}^n (p+1)^i$  if the  $\text{Datalog}^\circ$  program is linear. Finally, the convergence rate of a  $\text{Datalog}^\circ$  program is at most  $n$  if  $p = 0$ .

Thus the natural question left open by [10] was whether these upper bounds on the rate of convergence are (approximately) tight, and thus convergence can be exponential, or whether significantly better bounds are achievable.

## 1.2 Our Results

When a  $\text{Datalog}^\circ$  program over a semiring  $\mathcal{S}$  is linear, its ICO  $f : S^n \rightarrow S^n$  is a linear function of the form  $f(x) = A \otimes x \oplus b$ , where  $A$  is an  $n$  by  $n$  matrix with entries from  $S$ ,  $b$  is an  $n \times 1$  column vector with entries from  $S$ , and the scalar multiplication and addition are from  $\mathcal{S}$ . To simplify notations, we will use  $+$  and  $\cdot$  to denote the operations  $\oplus$  and  $\otimes$  respectively. Further, following the convention, we may omit  $\otimes$  if it is clear from the context. Then, we have

$$f(x) = Ax + b$$

► **Example 3.** For the APSP  $\text{Datalog}^\circ$  program,  $n$  is the number of edges in the graph,  $b_{(u,v)}$  is  $E(u,v)$ , and the matrix  $A$  would be:

$$A_{(u,v),(u,w)} = \begin{cases} E(v,w) & \text{if } u \neq v \\ 0 & \text{otherwise} \end{cases} \quad (8)$$

The stability index of  $f$  can easily be expressed in terms of the matrix  $A$  and vector  $b$ . Letting  $A^0 = I$  where  $I$  is the identity matrix, we have

$$f^{(k)}(x) = A^k x + A^{(k-1)}b \quad \text{where } A^{(k)} := \sum_{h=0}^{k-1} A^h.$$

Thus, the linear function  $f = Ax + b$  is *p*-stable if and only if  $A^{(p)}b = A^{(p+1)}b$ . Using the simple form of  $f$  for the linear case, we can rewrite Definition 2 into the following simpler form.

► **Observation 4** (Convergence Rate of a Linear  $\text{Datalog}^\circ$  Program).

- The convergence rate for a particular linear  $\text{Datalog}^\circ$  program, with associated matrix  $A$  and vector  $b$ , is the minimum natural number  $p$  such that  $A^{(p)}b = A^{(p+1)}b$ .
- The convergence rate for general linear  $\text{Datalog}^\circ$  programs over a semiring  $\mathcal{S}$  is then the maximum over all choices of  $A$  and  $b$ , of the convergence rate for that particular  $A$  and  $b$ .

## 11:6 On the Convergence Rate of Linear Datalog<sup>◦</sup> over Stable Semirings

Our first result is an asymptotically tight bound of  $\Theta(pn^3)$  on the rate of convergence for linear Datalog<sup>◦</sup> programs.

► **Theorem 5.** *Every linear Datalog<sup>◦</sup> program over a  $p$ -stable commutative semiring  $\mathcal{S}$  converges in  $O(pn^3)$  steps, where  $n$  is the total number of ground IDB atoms.*

The proof Theorem 5 can be found in Section 3, but we will give a brief overview of the main ideas of the proof here. Consider the complete  $n$ -vertex loop-digraph  $G$  where the edge from  $i$  to  $j$  is labeled with entry  $A_{i,j}$ . Then note that row  $i$  column  $j$  of  $A^{(h)}$  is the sum – over all walks  $W$  from  $i$  to  $j$  of length  $\leq h$  – of the product of the edge labels on the walk. We show that the summand corresponding to a walk  $W$  with more than  $h = \Omega(pn^3)$  hops doesn't change the sum  $A_{i,j}^{(h)}$ . We accomplish this by rewriting the summand corresponding to  $W$ , using the commutativity of multiplication in  $\mathcal{S}$ , as the product of a simple path and of multiple copies of at most  $n^2 - n$  distinct closed walks. We then note that, by the pigeon hole principle, one of these closed walks, say  $C$ , must have multiplicity greater than  $p$ . We then conclude that the summand corresponding to  $W$  will not change the sum  $A_{i,j}^{(h)}$  by appealing to the stability of the semiring element that is the product of the edges in  $C$ .

In section 4 we establish a matching lower bound by finding a graph  $G$  and a commutative semiring  $\mathcal{S}$  where the calculations in the upper bound are tight.

► **Theorem 6.** *For any  $p, n \geq 1$ , there is a linear Datalog<sup>◦</sup> program over a  $p$ -stable semiring  $\mathcal{S}$  that requires  $\Omega(pn^3)$  steps to converge.*

In the lower bound instance that establishes Theorem 6 the size of the ground set  $S$  of  $\mathcal{S}$  is of size  $\Theta((p+1)^{n^2})$ , so exponential in  $n$ . Thus one natural question is whether Datalog<sup>◦</sup> programs over semirings with subexponentially sized ground sets will converge more quickly. In section 5 we answer this question in the affirmative by showing that the rate of convergence of a Datalog<sup>◦</sup> program over a  $p$ -stable commutative semiring with  $L$  elements is  $O(pn \log L)$ .

► **Theorem 7.** *Every linear Datalog<sup>◦</sup> program over a  $p$ -stable commutative semiring that contains  $L$  elements in its ground set converges in  $O(pn \log L)$  steps.*

Let us explain the high level idea of the proof, with some simplifying assumptions, starting with the assumption that the stability of  $\mathcal{S}$  is  $p = 1$ . Again we think of  $A_{i,j}^{(k)}$  as the sum of products over walks  $W$  from  $i$  to  $j$  with at most  $k$  hops. Now consider a walk  $W$  from  $i$  to  $j$  consisting of  $\Omega(n \log L)$  hops. Since there are  $n$  vertices, there exists a vertex  $v$  such that there are at least  $\Omega(\log L)$  prefixes  $P_1, P_2, \dots, P_q$  of  $W$  ending at  $v$ . Let  $C_i$  be the closed walk starting and ending at  $v$  such that appending  $C_i$  to  $P_i$  produces  $P_{i+1}$ . Let us make the simplifying assumption that all of these closed walks  $C_i$  are distinct. The key observation is that if we delete any subset  $\mathcal{C}$  of these  $\Omega(\log L)$  closed walks from  $W$ , the result will still be a walk from  $i$  to  $j$ . Thus there are at least  $2^{\Omega(\log L)}$  walks from  $i$  to  $j$  that can be formed by deleting a subset  $\mathcal{C}$  of these closed walk. Since there are at most  $L$  distinct elements, by the pigeon hole principle, there must be some element  $e$  of  $\mathcal{S}$  such that there are two subsets  $\mathcal{C}, \mathcal{C}'$  where the product of the edges in them are the same. We then conclude by the stability of  $e$  that the summand corresponding to  $W$  does not change the sum  $A_{i,j}^{(k)}$ .

Finally in section 6 we establish a nearly matching lower bound by finding a graph  $G$  and a commutative semiring where the calculations in the upper bound are nearly tight.

► **Theorem 8.** *There are linear Datalog<sup>◦</sup> programs over a  $p$ -stable commutative semiring that contains  $L$  elements that require  $\Omega(pn \frac{\log L}{\log p})$  steps to converge.*

Finally in the appendix we consider a special case from [10], in which the commutative semiring  $\mathcal{S}$  is naturally ordered.



## 2 Preliminaries

In this section, we introduce the notation and terminology used throughout the paper.

► **Definition 9** (Semiring). A semiring [7] is a tuple  $\mathcal{S} = (S, \oplus, \otimes, 0, 1)$  where

- $\oplus$  and  $\otimes$  are binary operators on  $S$ ,
- $(S, \oplus, 0)$  is a commutative monoid, meaning  $\oplus$  is commutative and associative, and 0 is the identity for  $\oplus$ ,
- $(S, \otimes, 1)$  is a monoid, meaning  $\otimes$  is associative, and 1 is the identity for  $\otimes$ ,
- 0 annihilates every element  $a \in S$ , that is  $a \otimes 0 = 0 \otimes a = 0$ , and
- $\otimes$  distributes over  $\oplus$ .

A commutative semiring  $\mathcal{S} = (S, \oplus, \otimes, 0, 1)$  is a semiring where additionally  $\otimes$  is commutative.

If  $A$  is a set and  $p \geq 0$  a natural number, then we denote by  $\mathcal{B}_p(A)$  the set of bags (multiset) of  $A$  of size  $p$ , and  $\mathcal{B}_{\text{fin}}(A) := \bigcup_{p \geq 0} \mathcal{B}_p(A)$ . We denote bags as in  $\{\{a, a, a, b, c, c\}\}$ . Given  $\mathbf{x}, \mathbf{y} \in \mathcal{B}_{\text{fin}}(\mathbb{R}_+ \cup \infty)$ , define

$$\mathbf{x} \uplus \mathbf{y} := \text{bag union of } \mathbf{x}, \mathbf{y} \qquad \mathbf{x} + \mathbf{y} := \{\{u + v \mid u \in \mathbf{x}, v \in \mathbf{y}\}\}$$

► **Example 4.** For any multiset  $\mathbf{x} = \{\{x_0, x_1, \dots, x_n\}\}$ , where  $x_0 \leq x_1 \leq \dots \leq x_n$ , and any  $p \geq 0$ , define:

$$\min_p(\mathbf{x}) := \{\{x_0, x_1, \dots, x_{\min(p, n)}\}\}$$

In other words,  $\min_p$  returns the smallest  $p + 1$  elements of the bag  $\mathbf{x}$ . Then, for any  $p \geq 0$ , the following is a semiring:

$$\text{Trop}_p^+ := (\mathcal{B}_{p+1}(\mathbb{R}_+ \cup \{\infty\}), \oplus_p, \otimes_p, \mathbf{0}_p, \mathbf{1}_p)$$

where:

$$\begin{aligned} \mathbf{x} \oplus_p \mathbf{y} &\stackrel{\text{def}}{=} \min_p(\mathbf{x} \uplus \mathbf{y}) & \mathbf{0}_p &\stackrel{\text{def}}{=} \{\{\infty, \infty, \dots, \infty\}\} \\ \mathbf{x} \otimes_p \mathbf{y} &\stackrel{\text{def}}{=} \min_p(\mathbf{x} + \mathbf{y}) & \mathbf{1}_p &\stackrel{\text{def}}{=} \{\{0, \infty, \dots, \infty\}\} \end{aligned}$$

For example, if  $p = 2$  then  $\{\{3, 7, 9\}\} \oplus_2 \{\{3, 7, 7\}\} = \{\{3, 3, 7\}\}$  and  $\{\{3, 7, 9\}\} \otimes_2 \{\{3, 7, 7\}\} = \{\{6, 10, 10\}\}$ . The following identities are easily checked, for any two finite bags  $\mathbf{x}, \mathbf{y}$ :

$$\min_p(\min_p(\mathbf{x}) \uplus \min_p(\mathbf{y})) = \min_p(\mathbf{x} \uplus \mathbf{y}) \quad \min_p(\min_p(\mathbf{x}) + \min_p(\mathbf{y})) = \min_p(\mathbf{x} + \mathbf{y}) \quad (9)$$

Note that  $\text{Trop}_0^+$  is the natural “min-plus” semiring that we used in Example 2.

In the following fact about  $p$ -stable commutative semiring will be useful.

► **Proposition 10.** Given a  $p$ -stable commutative semiring  $\mathcal{S} = (S, \oplus, \otimes, 0, 1)$ , for any  $u \in S$ , we have  $pu = (p + 1)u$ , where  $pu$  here is shorthand for  $\oplus_{i=1}^p u$ .

**Proof.** This follows directly from the  $p$ -stability of 1, and the fact that  $1^2 = 1$ . ◀

Let us now explain the general procedure for creating a matrix  $A$  and a vector  $b$  from a linear Datalog<sup>o</sup> program  $Q$ . Each ground tuple for each IDB predicate  $R$  can be viewed as a variable, and ground tuples of EDB predicates can be viewed as constants. Then a Datalog<sup>o</sup> rule, where the head is IDB predicate  $R$ , can be converted into a collection of linear equations, one for each ground tuple of  $R$ . Since all rules that share IDB  $R$  as the head can be combined via  $\oplus$ , we can compactly rewrite the entire set of Datalog<sup>o</sup> rules as a collection of linear equations of the following form:



$$T_i \leftarrow \bigoplus_{j=1}^n (A_{ij} \otimes T_j) \oplus b_i$$

where  $T_1, T_2, \dots, T_n$  are the variables corresponding to ground tuples of IDB predicates. By using more familiar notation  $+$ ,  $\cdot$  in the lieu of  $\otimes, \oplus$ , we can model a linear Datalog<sup>o</sup> program  $Q$  by a linear function  $f : S^n \rightarrow S^n$  of the form:

$$f(x) = Ax + b$$

where  $n$  is the total number of ground tuples across all IDB predicates,  $x$  is an  $n$ -dimensional vector with entries from  $S$ ,  $A$  is an  $n$  by  $n$  matrix with entries from the  $S$ , and  $b$  is a dimensional column vector with entries from  $S$ . For some more examples of converting linear Datalog<sup>o</sup> programs into linear equations see [10].

### 3 Upper bounding the Convergence as a Function of the Matrix Dimension and the Semiring Stability

This section is devoted to proving Theorem 5, the main theorem of the paper. More specifically, we will show the following lemma. This lemma upper bounds the convergence of a  $p$ -stable semiring and implies Theorem 5.

► **Lemma 11.** *Let  $A$  be an  $n \times n$  matrix over a  $p$ -stable semiring  $S$ . Then  $A^{(k+1)} = A^{(k)}$ , where  $k = n(n^2 - n)(p + 2) + n - 1$ .*

Consider the complete  $n$ -vertex loop-digraph  $G$  where the edge from  $i$  to  $j$  is labeled with entry  $A_{i,j}$ . Then

$$A_{i,j}^h = \sum_{W \in \mathcal{W}_{i,j}^h} \Phi(W)$$

where  $\mathcal{W}_{i,j}^h$  is the collection of all  $h$ -hop walks from  $i$  to  $j$  in  $G$ , and

$$\Phi(W) = \prod_{(a,b) \in W} A_{a,b}$$

is the product off all the labels on all the directed edges in  $W$ . That is, row  $i$  column  $j$  of  $A^h$  is the sum over all  $h$ -hop walks  $W$  from  $i$  to  $j$  of the product of the labels on the walk. Similarly then,

$$A_{i,j}^{(h)} = \sum_{g=0}^h A_{i,j}^g = \sum_{g=0}^h \sum_{W \in \mathcal{W}_{i,j}^g} \Phi(W)$$

That is, row  $i$  column  $j$  of  $A^{(h)}$  the sum over all walks  $W$  from  $i$  to  $j$  with at most  $h$  hops of the product of the labels on the walk. Further,

$$A_{i,j}^{(h+1)} = A_{i,j}^{(h)} + A_{i,j}^{h+1} = A_{i,j}^{(h)} + \sum_{W \in \mathcal{W}_{i,j}^{h+1}} \Phi(W)$$

Our proof technique is to show that, by the  $p$ -stability of  $S$ , it must be the case that for each  $W \in \mathcal{W}_{i,j}^{p+1}$  it is the case that

$$A_{i,j}^{(k)} + \Phi(W) = A_{i,j}^{(k)}$$

The proof that  $A_{i,j}^{(k)} = A_{i,j}^{(k+1)}$  then immediately follows by applying this fact to each  $W \in \mathcal{W}_{i,j}^{k+1}$ .

Fix  $W = i, \dots, j$  to be an arbitrary walk in  $\mathcal{W}_{i,j}^{k+1}$ . Our next intermediate goal is to rewrite  $\Phi(W)$  using the commutativity of multiplication in  $S$  as the product of a simple path and at most  $n^2 - n$  multicycles. That is,

$$\Phi(W) = \Phi(P) \prod_{h=1}^{\ell} \Phi(C_h^{z_h})$$

where  $P$  is a simple path from  $i$  to  $j$  in  $G$ , each  $C_h$  is a simple cycle in  $W$  that is repeated  $z_h$  times, and  $\ell \leq n^2 - n$ . We accomplish this goal via the following tail-recursive construction. The recursion is passed a collection of edges, and a parameter  $h$ . Initially, the edges are those in  $W$  and  $h$  is set to zero.

**Recursive Construction.** The base case is if  $W$  is a simple path. In the base case the path  $P$  is set to  $W$  and  $\ell$  is set to 0. Otherwise:

- $h$  is incremented
- $C_h$  is set to be an arbitrary simple cycle in  $W$ .
- Let  $z_h$  be the minimum over all edges  $e \in C_h$  of the number of times that  $e$  is traversed in  $W$ .
- Let  $W'$  be the collection of edges in  $W$  except that  $z_h$  copies of every edge in  $C_h$  are removed.
- The construction then recurses on  $W'$  and  $h$ .

In Lemma 13 we show that a particular statement about  $W$  is invariant through the recursive construction. A proof Lemma 13 requires the following lemma. The proof of the following lemma (or at least, the techniques needed for a proof) can be found in most introductory graph theory texts, e.g. [20] Theorem 23.1.

► **Lemma 12.**

- A loop digraph  $G$  has a Eulerian walk from from a vertex  $i$  to a vertex  $j$ , where  $i \neq j$ , if and only if vertex  $i$  has out-degree one greater than its in-degree, vertex  $j$  has out-degree one less than its in-degree, every other vertex has equal in-degree and out-degree, and all of the vertices with nonzero degree belong to a single connected component of the underlying undirected graph.
- A loop digraph has an Eulerian cycle that includes a vertex  $i$  if and only if every vertex has equal in-degree and out-degree, vertex  $i$  has non-zero in-degree, and all of the vertices with nonzero degree belong to a single connected component of the underlying undirected graph.

► **Lemma 13.** Let  $W$  be a collection of edges that is passed at some point in the recursive construction. Let  $D_1, \dots, D_h$  be a partition of the edges of  $W$  with the property that if the edges in  $W$  were viewed as undirected, then the connected components would be  $D_1, \dots, D_h$ .

- If  $i \in D_f$  then  $D_f$  is a walk from  $i$  to  $j$ .
- If  $i \notin D_f$  then  $D_f$  is a Eulerian circuit.

**Proof.** The proof is by induction on the number of steps of the recursive construction. The statement is obviously true for the initial walk  $W$ , which is the base case. Now consider one step of the recursive construction. Removing copies of a cycle from  $W$  does not change the difference between the in-degree and out-degree of any vertex. Thus by Lemma 12 the only

## 11:10 On the Convergence Rate of Linear Datalog<sup>o</sup> over Stable Semirings

issue we need to consider is vertex  $i$  and vertex  $j$  possibly ending up in different connected components of  $W'$ . Let  $D_f$  be the connected component of  $W$  that contains  $i$  (which also contains  $j$  by the induction hypothesis), let  $D'_a$  be the connected component of  $W'$  that contains  $i$ , and let  $D'_b$  be the connected component of  $W'$  that contains  $j$ , where  $a \neq b$ . Then the walk in  $D_f$  from  $i$  to  $j$  must cross the cut (in either direction) formed by the vertices in  $D'_a$  an odd number of times. But the edges in  $C_h$  must cross the cut (in either direction) formed by the vertices in  $D'_a$  an even number of times. Thus there must be an edge in  $D_f$  minus  $z_h$  copies of  $C_h$  that must cross the cut (in either direction) formed by the vertices in  $D'_a$ . However, then this is a contradiction to  $D'_a$  be a connected component in  $W'$ . ◀

We now make a sequence of observations, that will eventually lead us to our proof of Lemma 11.

► **Observation 14.**

- $1 \leq \ell \leq n^2 - n$ .
- *The recursive construction terminates.*
- $\Phi(W) = \Phi(P) \prod_{h=1}^{\ell} \Phi(C_h^{z_h})$ .

**Proof.** As  $W$  contains  $k + 1 = n(n^2 - n)(p + 2) + n$  edges, then it must contain a simple cycle, which implies  $\ell \geq 1$ . Consider one iteration of our recursive construction. There must be a directed edge  $e \in C_h$  that appears exactly  $z_h$  times in  $W$ . Thus there are no occurrences of  $e$  in  $W'$ . Thus  $e$  can not appear in any future cycles, that is  $e \notin C_g$  for any  $g > h$ . The first observation then follows because there are at most  $n^2 - n$  different edges in  $G$ . The second observation is then an immediate consequence of the first observation, and the invariant established in Lemma 13. The third observation follows because no edges are ever lost or created in the recursive construction. ◀

► **Observation 15.** *There is a cycle  $C_s$ ,  $1 \leq s \leq \ell$ , such that  $z_s$  is at least  $p + 2$ .*

**Proof.** Since  $P$  is a simple path it contains at most  $n - 1$  edges. Thus  $W - P$  contains at least  $n(n^2 - n)(p + 2)$  edges. As any simple cycle contains at most  $n$  edges, and as there are at most  $\ell \leq n^2 - n$  cycles, then by applying the pigeon hole principle to the cycle decomposition of  $W$  we can conclude that there must be a cycle  $C_s$  that has multiplicity at least  $p + 2$ , that is  $z_s \geq p + 2$ . ◀

For convenience, consider renumbering the cycles so that  $s = 1$  where  $z_s \geq p + 1$ , which exists by Observation 15. For each  $h$  such that  $1 \leq h \leq p + 1$ , let define  $W_h$  be the collection of edges in  $W$  minus  $h$  copies of every edge in  $C_1$ .

► **Observation 16.** *The edges in each  $W_h$ ,  $1 \leq h \leq p + 1$  form a walk from  $i$  to  $j$ .*

**Proof.** As  $z_1 \geq p + 2$  and  $h \leq p + 1$ , every edge that appears in  $W$  also appears in  $W_h$ . So  $W_h$  has the same connectivity properties as  $W$ , and  $W_h$  has the same vertices with positive in-degree as does  $W$ . Also as  $C_1$  is a simple cycle, the difference between in-degree and out-degree for each vertex is the same in  $W_h$  as in  $W$ . Thus the result follows by appealing to Lemma 12. ◀

► **Observation 17.** *For each  $h$  such that  $1 \leq h \leq p + 1$  it is the case that*

$$\Phi(W_h) = \Phi(P) \Phi(C_1^{z_1 - h}) \prod_{f=2}^{\ell} \Phi(C_f^{z_f})$$

**Proof.** This follows directly from the definition of  $W_h$ . ◀

► **Observation 18.** For all  $h$  such that  $1 \leq h \leq p+1$ , we have  $W_h \in \bigcup_{f=0}^k \mathcal{W}_{i,j}^f$ . That is  $\Phi(W_h)$  appears as a term in  $A_{i,j}^{(k)}$ .

**Proof.** This follows because  $W$  has  $k+1$  edges and  $C_1$  is non-empty, so removing edges in  $C_i$  strictly decreases the number of edges. ◀

We are now ready to prove Lemma 11. By Observation 18 we know that each  $\Phi(W_h)$  is included in  $A^{(k)}$ , and thus there exists an element  $r$  in the semiring  $S$  such that

$$A_{i,j}^{(k)} = r + \sum_{h=1}^{p+1} \Phi(W_h)$$

Thus

$$\begin{aligned} A_{i,j}^{(k)} + \Phi(W) &= r + \sum_{h=1}^{p+1} \Phi(W_h) + \Phi(W) \\ &= r + \sum_{h=1}^{p+1} \left( \Phi(P) \Phi(C_1^{z_1-h}) \prod_{f=2}^{\ell} \Phi(C_f^{z_f}) \right) + \Phi(P) \prod_{f=1}^{\ell} \Phi(C_f^{z_f}) \\ &= r + \left( \Phi(P) \prod_{f=2}^{\ell} \Phi(C_f^{z_f}) \right) \left( \sum_{h=1}^{p+1} \Phi(C_1^{z_1-h}) + \Phi(C_1^{z_1}) \right) \end{aligned} \quad (10)$$

$$\begin{aligned} &= r + \left( \Phi(P) \prod_{f=2}^{\ell} \Phi(C_f^{z_f}) \right) \left( \Phi(C_1^{z_1-(p+1)}) \sum_{h=0}^{p+1} \Phi(C_1^h) \right) \\ &= r + \left( \Phi(P) \prod_{f=2}^{\ell} \Phi(C_f^{z_f}) \right) \left( \Phi(C_1^{z_1-(p+1)}) \sum_{h=0}^{p+1} [\Phi(C_1)]^h \right) \end{aligned} \quad (11)$$

$$\begin{aligned} &= r + \left( \Phi(P) \prod_{f=2}^{\ell} \Phi(C_f^{z_f}) \right) \left( \Phi(C_1^{z_1-(p+1)}) \sum_{h=0}^p [\Phi(C_1)]^h \right) \end{aligned} \quad (12)$$

$$\begin{aligned} &= r + \left( \Phi(P) \prod_{f=2}^{\ell} \Phi(C_f^{z_f}) \right) \left( \Phi(C_1^{z_1-(p+1)}) \sum_{h=0}^p \Phi(C_1^h) \right) \end{aligned}$$

$$\begin{aligned} &= r + \left( \Phi(P) \prod_{f=2}^{\ell} \Phi(C_f^{z_f}) \right) \left( \sum_{h=1}^{p+1} \Phi(C_1^{z_1-h}) \right) \end{aligned}$$

$$\begin{aligned} &= r + \sum_{h=1}^{p+1} \left( \Phi(P) \Phi(C_1^{z_1-h}) \prod_{f=2}^{\ell} \Phi(C_f^{z_f}) \right) \end{aligned}$$

$$\begin{aligned} &= r + \sum_{h=1}^{p+1} \Phi(W_h) = A_{i,j}^{(k)} \end{aligned}$$

The equality in line (10) follows from Observation 17. The equality in line (11) follows from the definition of  $\Phi$ . The key step in this line of equations is the equality in line (12), which follows from the stability of  $\Phi(C_1)$ . The rest of the equalities follow from basic algebraic properties of semirings, or by definition of the relevant term.

**4 A Lower Bound for Convergence Rate of Linear Datalog<sup>o</sup> Programs**

This section lower bounds the convergence rate of linear Datalog<sup>o</sup> programs using naive evaluation and establishes Theorem 6. In particular, this section will construct a semiring and a datalog program that requires  $\Omega(pn^3)$  iterations to converge. The section first constructs a semiring then defines a matrix  $A$  and finally the converge rate is bounded. We remark that we only show this lower bound holds for this specific semiring and matrix.

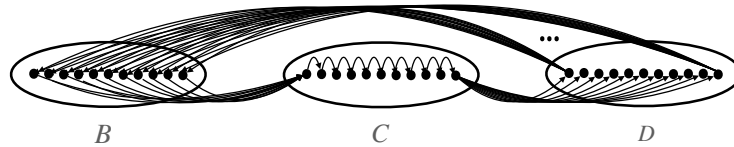
**4.1 Constructing the Semiring**

The ground set  $S$  consists of multi-sets of  $\{1, \dots, m\}$ ; later we will set  $m = \Theta(n^2)$ . To avoid confusion, we will refer to  $1, 2, \dots, m$  as items, which will be distinguished from the elements of  $S$ . An element in  $S$  can have up to  $p \geq 1$  copies of each item  $i \in [m]$ . Additionally, there is a special element  $\mathcal{O}$  in the semiring. Thus, the ground set  $S$  has  $(p + 1)^m + 1$  elements.

For a multiset  $A$  we denote the number of copies of item  $i$  in  $A$  as  $A_i$ . We now define the semiring operations. Consider two distinct elements (multisets)  $A$  and  $B$  in  $S \setminus \{\mathcal{O}\}$ . Define  $C = A + B$ , where  $C_i = \max\{A_i, B_i\}$  for all  $i \in [m]$ . Further,  $\mathcal{O} + A = A + \mathcal{O} = A$  for all  $A \in S$ . Multiplication  $C = A \cdot B$  is defined as follows:  $C_i = \min\{A_i + B_i, p\}$  for all  $i \in [m]$ . Further  $\mathcal{O} \cdot A = A \cdot \mathcal{O} = \mathcal{O}$  for all  $A \in S$ .

We establish that this is a semiring where  $\mathcal{O}$  is the additive identity and the empty multi-set  $\emptyset$  is the multiplicative identity (i.e. 1). By definition,  $\mathcal{O}$  is the annihilator element. We show in Appendix B that this is a commutative semiring. The proof easily follows from the definition of the semiring.

**4.2 Defining the Matrix  $A$**



This section defines the matrix  $A$ . To do so, we first define a graph  $G$ . Let  $G$  be a directed graph where the vertex set is the integers from 1 to  $n$  inclusive. For simplicity assume  $n$  is divisible by 3. The vertices are partitioned into 3 parts,  $B = \{1, \dots, n/3\}$ ,  $C = \{n/3 + 1, \dots, 2n/3\}$  and  $D = \{2n/3 + 1, \dots, n\}$ . There is a directed edge from each vertex in  $B$  to vertex  $n/3 + 1$ , there is a directed edge from vertex  $2n/3$  to each vertex in  $D$ , and there is a directed edge from each vertex in  $D$  to each vertex in  $B$ . Finally, all vertices in  $C$  are sequentially connected from  $n/3 + 1$  to  $2n/3$ , i.e., there is a directed edge from  $\tau$  to  $\tau + 1$  for all  $\tau \in [n/3 + 1, 2n/3 - 1]$ .

Index the edges by 1 through  $m$  and assign distinct labels (items) to them. So,  $m$  is exactly equal to the number of distinct items. Notice that  $m$  is  $\Theta(n^2)$ . If there is a directed edge  $(i, j)$  with label  $k$  in  $G$  then  $A_{i,j} = \{k\}$ . This is the element corresponding to the multiset with one copy of  $k$ . If there is no directed edge  $(i, j)$  in  $G$  then  $A_{i,j} = \mathcal{O}$ .

► **Lemma 19.** *The number of steps until convergence is  $\Omega(pn^3)$ .*

**Proof.** Note that there are  $|B| \cdot |D| = n^2/9$  edges from  $D$  to  $B$ . Consider a long walk, say from  $i := n/3 + 1$  to  $j := n/3 + 2$ . Observe that the walk must visit all edges within  $C$  (and the edge from  $2n/3$  to  $2n/3 + 1$ ) before visiting exactly one edge from  $D$  to  $B$ . Thus, to visit each

edge from  $D$  to  $B$  at least  $p$  times, the walk must have length at least  $p|B| \cdot |D| \cdot |C| = pn^3/27$ . Similarly, there is such a walk of length at most  $p|B| \cdot |D|(|C| + 2) + 1 \leq 4pn^3$ . Thus, we have shown that  $A^{(k)}$  includes the multiset  $Q$  that has  $p$  copies of each item when  $k \geq 4pn^3$ . Further, we have shown that  $A^{(k)}$  doesn't include  $Q$  when  $k < p(n/3)^3$ . This proves the lower bound on the convergence rate.  $\blacktriangleleft$

## 5 Bounding Convergence in Terms of the Semiring Ground Set Size

This section investigates the convergence rate with the assumption that the semiring has a ground set of size at most  $L$ . With this assumption, we can prove significantly better upper bounds. This section's goal is to prove Theorem 7. We prove the following lemma, which will immediately imply Theorem 7. As before, we let  $\Phi(W) = \prod_{e \in W} e$  for a walk  $W$ .

► **Lemma 20.** *Let  $A$  be an  $n$  by  $n$  matrix both a  $p$ -stable semiring  $S$  with a ground set consisting of  $L$  elements. Then  $A^{(k+1)} = A^{(k)}$ , where  $k = \lceil 8p(\lg L + 1)n \rceil + 1 = O(np \lg L)$ .*

**Proof.** Fix  $i, j \in [n]$ . Consider any  $W \in \mathcal{W}_{i,j}^{k+1}$  for the value of  $k$  stated in the lemma. To show the lemma it suffices to show  $\Phi(W) + A_{i,j}^{(k)} = A_{i,j}^{(k)}$ . By the pigeon hole principle, there must exist a vertex  $v$  visited at least  $8p(\lg L + 1) + 1$  times. We now conceptually cut  $W$  at visits to  $v$  to form a cycle decomposition of  $W$ . That is, we can write  $W$  as  $TC_1, \dots, C_hT'$  where  $T$  is a walk from  $i$  to  $v$  such that the only time it visits vertex  $v$  is on the last step, each  $C_i$  is a closed walk that includes vertex  $v$  exactly once, and  $T'$  is a walk from  $v$  to  $j$  that doesn't visit  $v$  again after initially leaving  $v$ . Note by the definition of vertex  $v$  it must be the case that  $h \geq 8p(\lg L + 1)$ . Let  $\mathcal{C} = \{C_f \mid 1 \leq f \leq h\}$  be the collection of cycles in this cycle decomposition.

We now partition  $\mathcal{C}$  into parts  $\mathcal{C}_1, \dots, \mathcal{C}_\ell, \mathcal{J}$  where for each closed walk  $C$  that has multiplicity  $m$  in  $\mathcal{C}$  there are  $2^f$  copies of  $C$  in  $\mathcal{C}_f$  and  $m - 2^f$  copies of  $C$  in  $\mathcal{J}$  where  $f = \lfloor \lg m \rfloor$ . For a collection  $\mathcal{D}$  of cycles, it will be convenient to use  $\Phi(\mathcal{D})$  to denote  $\Phi(\bigcup \mathcal{D})$ . Thus it then immediate that  $\Phi(W) = \Phi(T)\Phi(\mathcal{C})\Phi(T')\Phi(\mathcal{J})$ . Note that the cardinality of the multiset  $\bigcup_{f=1}^\ell \mathcal{C}_f$  is at least  $4p(\lg L + 1)$ .

We change  $\mathcal{C}$  repeatedly without changing  $\Phi(\mathcal{C})$ . When changing  $\mathcal{C}$  into  $\mathcal{C}'$ , we satisfy:

1.  $\Phi(\mathcal{C}) = \Phi(\mathcal{C}')$ .
2.  $N(\mathcal{C}) = N(\mathcal{C}')$ , where  $N(\mathcal{C})$  denotes the size of multi-set  $\mathcal{C}$ . So, a cycle  $C$  contributes to  $N(\mathcal{C})$  by the number of times it appears in  $\mathcal{C}$ . Further,  $\mathcal{C}'$  has no more edges in total than  $\mathcal{C}$  when we count 1 for each edge in one cycle appearance, i.e.,  $\sum_{C' \in \mathcal{C}'} |C'| \leq \sum_{C \in \mathcal{C}} |C|$ .
3. Consider  $\langle \dots, N_3(\mathcal{C}'), N_2(\mathcal{C}'), N_1(\mathcal{C}') \rangle$  and  $\langle \dots, N_3(\mathcal{C}), N_2(\mathcal{C}), N_1(\mathcal{C}) \rangle$ . The first vector dominates the second lexicographically. Here  $N_\ell(\mathcal{C})$  denotes the number of cycles in  $\mathcal{C}$  of exponent  $2^\ell$ .
4. When we terminate,  $N_\ell(\mathcal{C}) \leq 2 \lfloor \lg L + 1 \rfloor$  for all  $\ell \leq \lfloor \lg p \rfloor$ .
5. Every cycle in  $\mathcal{C}'$  also appears in  $\mathcal{C}$ .

We now describe the transformation from  $\mathcal{C}$  into  $\mathcal{C}'$ . Consider the smallest  $\ell$  such that  $N_\ell > 2 \lfloor \lg L + 1 \rfloor$ . Consider every subset of cardinality  $\lfloor \lg L + 1 \rfloor$ , that consists of cycles of exponent  $2^\ell$ ; so they are in  $\mathcal{C}_\ell$ . The number of such subsets is at least  $\binom{2 \lfloor \lg L + 1 \rfloor}{\lfloor \lg L + 1 \rfloor} > 2^{\lfloor \lg L \rfloor} = L$ . Since the semiring has at most  $L$  distinct elements, there must exist distinct subsets  $A$  and  $B$  of cycles of exponent  $2^\ell$  such that  $|A| = |B|$  and  $\Phi(A) = \Phi(B)$ . Assume wlog that  $B$ 's cycles have no more edges in total than  $A$ 's cycles. Then, we replace  $A$  with  $B$  in  $\mathcal{C}$  and let  $\mathcal{C}'$  be  $\mathcal{C}$  after this change.

It is easy to see that  $\Phi(\mathcal{C}) = \Phi(\mathcal{C}')$ . This is because we replaced  $A$  with  $B$  such that  $\Phi(A) = \Phi(B)$ . More formally,  $\Phi(\mathcal{C}) = \prod_{\ell'} \Phi(\mathcal{C}_{\ell'}) = (\Phi(\mathcal{C}_\ell \setminus A)\Phi(A)) \prod_{\ell' \neq \ell} \Phi(\mathcal{C}_{\ell'}) = (\Phi(\mathcal{C}_\ell \setminus A)\Phi(B)) \prod_{\ell' \neq \ell} \Phi(\mathcal{C}_{\ell'}) = \Phi(\mathcal{C}')$ . The second property is also obvious because when

## 11:14 On the Convergence Rate of Linear Datalog<sup>o</sup> over Stable Semirings

we replace  $A$  with  $B$  in the change, we ensured  $|A| = |B|$ , which implies that the multiset size remains unchanged. Also, it is immediate that the total number of edges don't increase as  $B$  has no more edges in total than  $A$ . The fourth property, the termination condition, is immediate. The fifth property is also immediate since we do not create a new cycle when replacing  $A$  with  $B$ .

To see the third property, before replacing  $A$  with  $B$ , we had  $A \cup B$  in  $\mathcal{C}_\ell$ , and their exponent was  $2^\ell$ . After the replacement, all cycles in  $B \setminus A$  come to have exponent  $2 \cdot 2^\ell = 2^{\ell+1}$ , the cycles in  $A \setminus B$  disappear from  $\mathcal{C}_\ell$ , and those in  $A \cap B$  remain unchanged. Note that every cycle remains to have an exponent that is a power of two. Therefore, we have  $N_{\ell+1}(\mathcal{C}') > N_{\ell+1}(\mathcal{C})$ , and  $N_{\ell'}(\mathcal{C}') = N_{\ell'}(\mathcal{C})$  for all  $\ell' \geq \ell + 2$ .

Observe that due to the second property and the third, the process must terminate. Thus, starting from  $\mathcal{C}$ , at the termination we have  $\mathcal{C}'$  that satisfies the first, second, and fourth properties. We know  $N(\mathcal{C}') = N(\mathcal{C}) > 4p(\lg L + 1)$ . Further, we know that cycles of exponent at most  $p$  contribute to  $N(\mathcal{C}')$  by at most  $2(\lg L + 1)(1 + 2 + 4 + \dots + 2^{\lfloor \lg p \rfloor}) < 4p(\lg L + 1)$ . Thus, there must exist a cycle in  $\mathcal{C}'$  of exponent greater than  $p$ . Let  $C$  denote the cycle. So, we have shown that

$$\Phi(W) = \Phi(T)\Phi(\mathcal{C}')\Phi(T') = \Phi(T)\Phi(C)^q\Phi(\mathcal{C}' \setminus C^q)\Phi(T'),$$

where  $q \geq p + 1$ . Here  $\mathcal{C}' \setminus C^q$  implies the resulting collection of cycles we obtain after removing  $q$  copies of  $C$  from  $\mathcal{C}'$ . Now consider walk  $W_{q'}$  that concatenates  $T$ ,  $q'$  copies of  $C$ ,  $\mathcal{C}' \setminus C^{q'}$ , and  $T'$ . Here, the walk starts with  $T$  and ends with  $T'$ , and the cycles can be placed in an arbitrary order. This is because  $T$  is a walk from  $i$  to  $v$ , and all the cycles in  $\mathcal{C}'$  start from  $v$  and end at  $v$  – due to the fifth property – and  $T'$  is a walk from  $v$  to  $j$ . Further, they are all shorter than  $W$  because  $W_q$  is no longer than  $W$  due to the second property. Therefore,  $W_0, W_1, \dots, W_{q-1} \in \mathcal{W}_{i,j}^{(k)}$ . Thus, thanks to  $p$ -stability, we have  $\Phi(W) + A_{i,j}^{(k)} = A_{i,j}^{(k)}$  as desired. ◀

Although we gave the full proof of Theorem 7, to convey intuition better, we also give some warm-up analyses in Appendix C by giving a looser bound for the general case and subsequently by considering a special case of  $p = 1$ .

### 6 Lower Bounds on Convergence in Terms of the Semiring Ground Set Size

This section constructs a lower bound of the convergence rate in terms of the size of the ground set. The goal is to show Theorem 8, which is implied by the following lemma.

► **Lemma 21.** *There exists an idempotent semiring on  $L$  elements and a matrix  $A$  of size  $n$  by  $n$  that requires  $\Omega(nL)$  steps to converge to a fixed point.*

**Proof.** Consider a semiring that whose all powers of 2 from 1 to  $2^L$  and the value 0. The value of  $2^L$  is the largest value. Summation  $A$  and  $B$  in the semiring is  $\min\{A \cdot B, 2^L\}$ , where  $\cdot$  is standard multiplication. Addition is standard maximum. By definition, addition is idempotent ensuring the semiring is idempotent.

The matrix  $A$  corresponds to a computation graph with a cycle of length  $n$ . All edges that are not in cycles are labeled 0. All edges of the cycle are labeled 1, the identity in standard multiplication, except for one edge, which is 2. Notice that multiplying all edges of the cycle  $i$  times results in the symbol  $2^i$ .

The cycle needs to be traversed  $L$  times to reach  $2^L$ . The walk is of length  $\Theta(nL)$ . ◀



► **Lemma 22.** *There exists a matrix  $A$  of size  $n$  by  $n$  which requires  $\Omega(np^{\frac{\log L}{\log p}})$  steps to find a fixed point over a semiring of  $L$  elements that is  $p$ -stable.*

**Proof.** Consider the following semiring. The  $L$  elements are over vectors of  $\ell$  dimensions. Each position can be  $0, 1, 2, \dots, p$ . Consider any two elements  $x$  and  $y$ . Let  $x_i$  and  $y_i$  be the  $i$ th dimension of  $x$  and  $y$ , respectively. The addition operation on  $x$  and  $y$  returns whichever among  $x$  and  $y$  are lexicographically larger. Multiplication of  $x$  and  $y$  produces the vector  $z$  where  $z_i = \min\{x_i + y_i, p\}$ .

We first claim that this is a semiring. Notice that the all 0 vector is a monoid for multiplication. The vector of all  $p$ 's is the multiplicative identity. The only case that is non-obvious is that multiplication distributes over addition. Consider three vectors  $a$ ,  $b$  and  $c$ . Consider the expression  $c \cdot (a + b)$ . We aim to show that this is equal to  $c \cdot a + c \cdot b$ . Without loss of generality say  $a$  is lexicographically bigger than  $b$ . Then we have that  $c \cdot (a + b) = c \cdot a$  because  $a + b = a$  using that  $a$  is lexicographically bigger than  $b$ . Similarly,  $c \cdot a + c \cdot b = c \cdot a$  because multiplication is standard addition and  $a$  is lexicographically bigger than  $b$ .

The matrix  $A$  corresponds to the following graph. There are two special nodes  $a$  and  $b$ . They are connected by  $\ell$  one-hop path using two edges from  $a$  to  $b$ . The  $k$ th path goes via a node  $c_k$ . The edge from  $a$  to  $c_k$  is labeled with the 0 vector. The edge from  $c_k$  to  $b$  is labeled with a vector of all 0s except a 1 in dimension  $k$ . Additionally,  $b$  is connected to  $a$  via a directed path of length  $n - \ell - 2$ . The edges of this path are all labeled with the all 0 vector.

To collect the vector consisting of  $p$  in each dimension, one needs to walk a cycle starting at  $a$  at least  $p\ell$  times. To see why, notice multiplication of the edges corresponding to a single time-around cycle increases a single dimension by at most one. Moreover, addition's definition ensures the final output is the vector that is lexicographically the biggest among each walk. Each cycle is of length  $\Omega(n)$ . The length of the walk required is  $\Omega(p\ell n)$ . Setting  $\ell = \frac{\log L}{\log p}$  gives the lemma by noting that  $L = p^\ell$  is the number of elements. ◀

## 7 Related Work

If the semiring is naturally ordered<sup>2</sup>, then the least fixpoint of a  $\text{Datalog}^\circ$  program is the least fixed point of  $f$  under the same partial order extended to  $S^n$  componentwise. This is the *least fixpoint* semantics of a  $\text{Datalog}^\circ$  program. The naïve evaluation algorithm for evaluating  $\text{Datalog}$  programs extends naturally to evaluating  $\text{Datalog}^\circ$  programs: starting from  $\mathbf{x} = 0^n$ , we repeatedly apply  $f$  to  $\mathbf{x}$  until a fixpoint is reached  $\mathbf{x} = f(\mathbf{x})$ . The core semiring of a POPS is naturally ordered. Thus, we can find the least fixpoint of a  $\text{Datalog}^\circ$  program by applying the naïve evaluation algorithm [10].

Computing the least fixpoint solution to a recursive  $\text{Datalog}^\circ$  program boils down to solving fixpoint equations over semirings. In particular, we are given a multi-valued polynomial function  $f : S^n \rightarrow S^n$  over a commutative semiring, and the problem is to compute a (pre-) fixpoint of  $f$ , i.e. a point  $x \in S^n$  where  $x = f(x)$ . As surveyed in [10], this problem was studied in a very wide range of communities, such as in automata theory [12], program analysis [4, 16], and graph algorithms [3, 14, 15] since the 1970s. (See [7, 8, 13, 17, 21] and references thereof).

When  $f = Ax + b$  is linear, as shown in the paper  $f^{(k)}(x) = A^{(k-1)}b$  and thus at fixpoint the solution is  $A^{(\omega)}b = \lim_{k \rightarrow \infty} A^{(k-1)}b$ , interpreted as a formal power series over the semiring. If there is a finite  $k$  for which  $A^{(k)} = A^{(k+1)}$ , then it is easy to see that

<sup>2</sup>  $S$  is naturally ordered if the relation  $x \preceq_S y$  defined as  $\exists z : x \oplus z = y$  is a partial order.

$A^{(\omega)} = A^{(k)}$ . The problem of computing  $A^{(\omega)}$  is called the *algebraic path problem* [17], which unifies many problems such as transitive closure [19], shortest paths [5], Kleene’s theorem on finite automata and regular languages [11], and continuous dataflow [4, 9]. If  $A$  is a real matrix, then  $A^{(\omega)} = I + A + A^2 + \dots$  is exactly  $(I - A)^{-1}$ , if it exists [2, 6, 18].

There are several classes of solutions to the algebraic path problem, which have pros and cons depending on what we can assume about the underlying semiring (whether or not there is a closure operator, idempotency, natural orderability, etc.). We refer the reader to [7, 17] for more detailed discussions.

---

## References

- 1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995. URL: <http://webdam.inria.fr/Alice/>.
- 2 R. C. Backhouse and B. A. Carré. Regular algebra applied to path-finding problems. *J. Inst. Math. Appl.*, 15:161–186, 1975.
- 3 Bernard Carré. *Graphs and networks*. The Clarendon Press, Oxford University Press, New York, 1979. Oxford Applied Mathematics and Computing Science Series.
- 4 Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In Robert M. Graham, Michael A. Harrison, and Ravi Sethi, editors, *Conference Record of the Fourth ACM Symposium on Principles of Programming Languages, Los Angeles, California, USA, January 1977*, pages 238–252. ACM, 1977. doi:10.1145/512950.512973.
- 5 Robert W. Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5(6):345, 1962. doi:10.1145/367766.368168.
- 6 M. Gondran. Algèbre linéaire et cheminement dans un graphe. *Rev. Française Automat. Informat. Recherche Opérationnelle Sér. Verte*, 9(V-1):77–99, 1975.
- 7 Michel Gondran and Michel Minoux. *Graphs, dioids and semirings*, volume 41 of *Operations Research/Computer Science Interfaces Series*. Springer, New York, 2008. New models and algorithms.
- 8 Mark W. Hopkins and Dexter Kozen. Parikh’s theorem in commutative kleene algebra. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*, pages 394–401. IEEE Computer Society, 1999. doi:10.1109/LICS.1999.782634.
- 9 John B. Kam and Jeffrey D. Ullman. Global data flow analysis and iterative algorithms. *J. ACM*, 23(1):158–171, 1976. doi:10.1145/321921.321938.
- 10 Mahmoud Abo Khamis, Hung Q. Ngo, Reinhard Pichler, Dan Suciu, and Yisu Remy Wang. Convergence of datalog over (pre-) semirings. In Leonid Libkin and Pablo Barceló, editors, *PODS ’22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*, pages 105–117. ACM, 2022. doi:10.1145/3517804.3524140.
- 11 S. C. Kleene. Representation of events in nerve nets and finite automata. In *Automata studies*, Annals of mathematics studies, no. 34, pages 3–41. Princeton University Press, Princeton, N. J., 1956.
- 12 Werner Kuich. Semirings and formal power series: their relevance to formal languages and automata. In *Handbook of formal languages, Vol. 1*, pages 609–677. Springer, Berlin, 1997. doi:10.1007/978-3-642-59136-5\_9.
- 13 Daniel J. Lehmann. Algebraic structures for transitive closure. *Theor. Comput. Sci.*, 4(1):59–76, 1977. doi:10.1016/0304-3975(77)90056-1.
- 14 Richard J. Lipton, Donald J. Rose, and Robert Endre Tarjan. Generalized nested dissection. *SIAM J. Numer. Anal.*, 16(2):346–358, 1979. doi:10.1137/0716027.
- 15 Richard J. Lipton and Robert Endre Tarjan. Applications of a planar separator theorem. *SIAM J. Comput.*, 9(3):615–627, 1980. doi:10.1137/0209046.
- 16 Flemming Nielson, Hanne Riis Nielson, and Chris Hankin. *Principles of program analysis*. Springer-Verlag, Berlin, 1999. doi:10.1007/978-3-662-03811-6.

- 17 Günter Rote. Path problems in graphs. In *Computational graph theory*, volume 7 of *Comput. Suppl.*, pages 155–189. Springer, Vienna, 1990. doi:10.1007/978-3-7091-9076-0\_9.
- 18 Robert E. Tarjan. Graph theory and gaussian elimination, 1976. J.R. Bunch and D.J. Rose, eds.
- 19 Stephen Warshall. A theorem on boolean matrices. *J. ACM*, 9(1):11–12, 1962. doi:10.1145/321105.321107.
- 20 Robin J. Wilson. *Introduction to Graph Theory*. Prentice Hall/Pearson, New York, 2010.
- 21 U. Zimmermann. Linear and combinatorial optimization in ordered algebraic structures. *Ann. Discrete Math.*, 10:viii+380, 1981.

## A

 Convergence of Naturally Ordered Semirings

► **Definition 23** (Natural Order). *In any (pre-)semiring  $\mathcal{S}$ , the relation  $x \preceq_{\mathcal{S}} y$  defined as  $\exists z : x \oplus z = y$ , is a preorder, which means that it is reflexive and transitive, but it is not anti-symmetric in general. When  $\preceq_{\mathcal{S}}$  is anti-symmetric, then it is a partial order, and is called the natural order on  $\mathcal{S}$ ; in that case we say that  $\mathcal{S}$  is naturally ordered.*

For simplicity, we may use  $\leq$  in lieu of  $\preceq$ . We naturally extend the ordering to vectors and matrices: for two vectors  $\mathbf{v}, \mathbf{w} \in \mathcal{S}^n$ , we have  $\mathbf{v} \leq \mathbf{w}$  iff  $v_i \leq w_i$  for all  $i \in [n]$ . Similarly, for two matrices  $A$  and  $B$  (which includes vectors)  $A \leq B$  means that componentwise each entry in  $A$  is at most the entry in  $B$ , that is for all  $i$  and  $j$  it is the case that  $A_{i,j} \leq B_{i,j}$ .

Here we take  $k$  to be the longest chain in the natural order.

► **Theorem 24.** *Every linear  $\text{Datalog}^\circ$  program over a  $p$ -stable naturally-ordered commutative semiring with maximum chain size  $k$  converges in  $O(kn)$  steps.*

► **Theorem 25.** *There are linear  $\text{Datalog}^\circ$  programs over a  $p$ -stable naturally-ordered commutative semiring with maximum chain size  $k$  that require in  $\Omega(kn)$  steps to converge.*

This section considers bounds in terms of the longest chain in the partial order of a naturally ordered semiring. Recall that natural ordering means the following: for two elements  $a$  and  $b$   $a \leq b$  if and only if there exists a  $c$  such that  $a + c = b$ . Let  $L$  be the length of the longest chain in this partial order. We seek to bound the convergence rate in terms of  $n$  and  $L$ .

► **Lemma 26.** *Consider a naturally ordered semiring where  $L$  is the length of the longest chain in the partial order. Let  $A$  be an  $n \times n$  matrix. Convergence must occur within  $nL$  steps.*

**Proof.** Consider  $A^{(k)}x$  as  $k$  increases for any fixed  $x$ . If there is a  $k \leq nL$  such that  $A^{(k)}x = A^{(k+1)}x$  then convergence has been reached within the desired number of steps. Otherwise when  $A^{(k)}x \neq A^{(k+1)}x$  there exists an  $i$  such that dimension  $i$  in  $A^{(k+1)}x$  is strictly greater than dimension  $i$  in  $A^{(k)}x$ . This can only occur  $L$  times for each  $i$  by definition of the partial order. Knowing that there are at most  $n$  dimensions in  $A^{(k)}x$ , the lemma follows. ◀

► **Lemma 27.** *There exists a naturally ordered semiring where  $L$  is the length of the longest chain in the partial order and a  $n$  by  $n$  matrix where convergence requires  $\Omega(nL)$  steps.*

**Proof.** Consider the following semiring. The semiring is on the set of integers  $0, 1, 2, \dots, L$  and a special element  $\mathcal{O}$ . Here, the additive identity is  $\mathcal{O}$  and the multiplicative identity is  $0$ . Consider two elements  $a$  and  $b$  that are not  $\mathcal{O}$ . Define the addition and multiplication of  $a$  and  $b$  to be equal to  $\min\{a + b, L\}$ . Define  $a$  multiplied by  $\mathcal{O}$  to be  $\mathcal{O}$  for any  $a$  and  $a$  added to  $\mathcal{O}$  to be  $a$  for any  $a$ . Intuitively, addition and multiplication act as standard addition capped at  $L$ , except for the special  $\mathcal{O}$  element.

## 11:18 On the Convergence Rate of Linear Datalog<sup>o</sup> over Stable Semirings

Consider the following graph corresponding to a  $n \times n$  matrix  $A$ . There is a cycle on  $n$  nodes. Order the edges from 1 to  $n$ . Each edge is labeled 0 except the edge from 1 to 2, which is labeled as 1. Traversing the cycle  $k$  times and multiplying the labels of the edges returns  $\min\{k, L\}$ . It takes a walk of length  $\Omega(nL)$  to reach the element  $L$ . ◀

### B Missing Proof from Section 4.1

We show that the semiring we defined in Section 4.1 is indeed a (commutative) semiring.

**Monoid  $(S, +)$  with identity  $\mathcal{O}$ :**

- $A + \mathcal{O} = \mathcal{O} + A = A$ . This follows from the definition.
- $(A + B) + C = A + (B + C)$ . When  $A, B, C \neq \mathcal{O}$ , this is equivalent to showing  $\max\{\max\{A_i, B_i\}, C_i\} = \max\{A_i, \max\{B_i, C_i\}\}$  for all  $i \in [m]$ , which follows from max being commutative. If  $A, B$ , or  $C$  is  $\mathcal{O}$ , then it is easy to check that it holds true.

**Monoid  $(S, \cdot)$  with identity  $\emptyset$ :**

- $A \cdot \emptyset = \emptyset \cdot A = A$ . If  $A \neq \mathcal{O}$ , this is immediate from the definition. If  $A = \mathcal{O}$ , again by definition,  $\mathcal{O} \cdot \emptyset = \emptyset \cdot \emptyset = \mathcal{O}$ .
- $(A \cdot B) \cdot C = A \cdot (B \cdot C)$ . When  $A, B, C \neq \mathcal{O}$ , it is an easy exercise to see  $((A \cdot B) \cdot C)_i = (A \cdot (B \cdot C))_i = \min\{A_i + B_i + C_i, p\}$ . If  $A, B$  or  $C$  is  $\mathcal{O}$ , both sides become  $\mathcal{O}$ .

**Commutative:**

- $A + B = B + A$ . If  $A, B \neq \mathcal{O}$ , we have  $(A + B)_i = \max\{A_i, B_i\} = (B + A)_i$ . Otherwise, it is immediate from the definition of  $\mathcal{O}$ .
- $A \cdot B = B \cdot A$ . We also show that the multiplication is also commutative. If  $A, B \neq \mathcal{O}$ , we have  $(A \cdot B)_i = \min\{A_i + B_i, p\} = \min\{B_i + A_i, p\} = (B \cdot A)_i$ . Otherwise  $A \cdot B = B \cdot A = \mathcal{O}$  from the definition.

**$\mathcal{O}$  is an multiplicative annihilator:** We have  $\mathcal{O} \cdot A = A \cdot \mathcal{O} = \mathcal{O}$  for all  $A \in S$  from the definition.

**Distributive:**

- $A \cdot (B + C) = A \cdot B + A \cdot C$ . Assume that  $A, B, C \neq \mathcal{O}$  since otherwise it is straightforward to see that it holds true. We then have,

$$\begin{aligned} (A \cdot (B + C))_i &= \min\{A_i + \max\{B_i, C_i\}, p\} = \min\{\max\{A_i + B_i, A_i + C_i\}, p\} \\ &= \max\{\min\{A_i + B_i, p\}, \min\{A_i + C_i, p\}\} = \max\{(A \cdot B)_i, (A \cdot C)_i\} \\ &= (A \cdot B + A \cdot C)_i \end{aligned}$$

- $(B + C) \cdot A = B \cdot A + C \cdot A$ . The proof is symmetric.

### C Warm-up for Proof of Theorem 7

In Section 5 we gave the full proof of Theorem 7, which gives an upper bound of  $O(np \log L)$  on the convergence rate when the underlying semiring has a ground set of size at most  $L$ .

To convey intuition better of the analysis, we give two warm-up proofs. Our first warm-up is giving a looser bound on the convergence rate. The proof makes use of the fact that a sufficiently long walk must visit the same vertex many times with the same product value. Here, we think of a prefix of the walk as a product of edges on the prefix. This prefix evaluates to an element of the semiring.

► **Lemma 28.** *Let  $A$  be an  $n$  by  $n$  matrix over a  $p$ -stable semiring on a ground set  $S$  consisting of  $L$  elements. Then  $A^{(k+1)} = A^{(k)}$ , where  $k = npL$ .*

**Proof.** Fix  $i, j \in [n]$ . Consider any  $W \in \mathcal{W}_{i,j}^{k+1}$ . To show the lemma it suffices to show  $\Phi(W) + A_{i,j}^{(k)} = A_{i,j}^{(k)}$ . Let  $W_h$  be the prefix of  $W$  of length  $h$ . Let  $v(W_h)$  denote the ending point of  $W_h$ . Consider all pairs  $(\Phi(W_h), v(W_h))$ ,  $h \in [k+1]$ . Since these tuples are subsets of  $S \times [n]$  and  $|S| = L$ , due to the pigeonhole principle, there must exist  $H \subseteq [k+1]$  of size  $p+1$  such that  $(\Phi(W_h), v(W_h))$  is the same tuple for all  $h \in H$ . By renaming we can represent the prefixes as  $X_1, X_1X_2, X_1X_2X_3, \dots, X_1X_2X_3 \dots X_{p+1}$ .

For some  $T$  (possibly empty), we have  $W = X_1X_2X_3 \dots X_{p+1}T$ . By definition  $\Phi(X_1) = \Phi(X_1X_2) = \dots = \Phi(X_1X_2 \dots X_{p+1})$ .

Thus,  $\Phi(W) = \Phi(X_1X_2X_3 \dots X_{p+1}T) = \Phi(X_1X_2X_3 \dots X_pT) \dots = \Phi(X_1T)$ , This uses the fact that  $X_1T, X_1X_2T, \dots, X_1X_2X_3 \dots X_pT$  all are walks from  $i$  to  $j$  since  $X_1, X_1X_2, \dots, X_1X_2 \dots X_{p+1}$  all end where  $T$  starts. Further, all walks  $X_1T, X_1X_2T, \dots, X_1X_2X_3 \dots X_pT$  are strictly shorter than  $W$ . This implies that  $\Phi(W)$  appears at least  $p$  times in  $A_{i,j}^{(k)}$ . Using Proposition 10, we conclude  $\Phi(W) + A_{i,j}^{(k)} = A_{i,j}^{(k)}$  as desired.  $\blacktriangleleft$

Next we consider the special case of  $p = 1$ . In this case we give an exponential improvement over what we showed in the previous lemma. The key idea is the following. Previously we identified  $p$  disjoint cycles  $X_2, X_3, \dots, X_{p+1}$  that share the same starting and ending vertex from a long walk  $W$  in  $\mathcal{W}_{i,j}^{k+1}$ . Then, by removing them sequentially we were able to obtain  $p$  copies of the same element that have already appeared; thus adding  $W$  (or more precisely  $\Phi(W)$ ) doesn't change  $A_{i,j}^{(k)}$ . Now we would like to make the same argument with an exponentially smaller number of cycles. Roughly speaking, we will identify  $\Theta(\lg L)$  such cycles and find  $2^{\Theta(\lg L)}$  walks by combining subsets of them. That is, the key idea is that we find more walks with the same product from far fewer cycles.

**► Lemma 29.** *Let  $A$  be an  $n$  by  $n$  matrix both over a 1-stable semiring  $S$  with a ground set consisting of  $L$  elements. Then  $A^{(k+1)} = A^{(k)}$ , where  $k = O(n \lg L)$ .*

**Proof.** As before, fix  $i, j \in [n]$ . Consider any  $k \geq \lceil 2 \lg L \rceil n$ . For any  $W \in \mathcal{W}_{i,j}^{k+1}$  we show  $\Phi(W) + A_{i,j}^{(k)} = A_{i,j}^{(k)}$ . Since there are  $n$  vertices, the walk must visit some vertex at least  $\lceil 2 \lg L \rceil + 1$  times. Formally, we can decompose  $W$  into

$$W = TC_1C_2 \dots C_H T' \quad (13)$$

where  $T, TC_1, TC_1C_2, \dots, TC_1C_2 \dots C_H$  all end at the same vertex  $v$ , and  $H = \lceil 2 \lg L \rceil$ . Note that all the cycles (or closed walks)  $C_1, C_2, \dots, C_H$  start from  $v$  and end at the same vertex  $v$ . It is plausible that some of them are identical.

For a subset  $A$  of  $[H]$  we let  $\hat{\Phi}(A) := \prod_{h \in A} \Phi(C_h)$ . Since there are  $2^{|H|}$  subsets of  $[H]$  and  $2^H > L$ , there must exist  $A, B \subseteq [H]$  such that  $A \neq B$  and  $\hat{\Phi}(A) = \hat{\Phi}(B)$ . Assume wlog that  $B \setminus A \neq \emptyset$ . Thus we know

$$\hat{\Phi}(A \cap B) \hat{\Phi}(A \setminus B) = \hat{\Phi}(A \cap B) \hat{\Phi}(B \setminus A) \quad (14)$$

We can then show,

$$\begin{aligned} \Phi(W) &= \Phi(T) \hat{\Phi}([H]) \Phi(T') \quad [\text{Eqn. 13}] \\ &= \Phi(T) \hat{\Phi}(A \cap B) \hat{\Phi}(A \setminus B) \hat{\Phi}(B \setminus A) \hat{\Phi}([H] \setminus (A \cup B)) \Phi(T') \\ &= \Phi(T) \hat{\Phi}(A \cap B) (\hat{\Phi}(B \setminus A))^2 \hat{\Phi}([H] \setminus (A \cup B)) \Phi(T') \quad [\text{Eqn. 14}] \end{aligned}$$

Consider a walk  $W'$  that starts with  $T$ , has  $C_h$  for each  $h \in (A \cap B) \cup (B \setminus A) \cup ([H] \setminus (A \cup B)) = [H] \setminus (A \setminus B)$  and ends with  $T'$ . Similarly, consider a walk  $W''$  that starts with  $T$ , has  $C_h$  for each  $h \in (A \cap B) \cup ([H] \setminus (A \cup B))$  and ends with  $T'$ . Note that  $W$  and  $W'$  are different

## 11:20 On the Convergence Rate of Linear Datalog<sup>o</sup> over Stable Semirings

since  $B \setminus A \neq \emptyset$ . Further they are walks from  $i$  to  $j$  since every cycle  $C_h$ ,  $h \in [H]$  starts from and ends at the same vertex  $v$ . Further, both walks are shorter than  $W$ , and therefore are in  $\mathcal{W}_{i,j}^{(k)}$ . Since we have  $\Phi(W') = \Phi(T)\hat{\Phi}(A \cap B)\hat{\Phi}(B \setminus A)\hat{\Phi}([H] \setminus (A \cup B))\Phi(T')$  and  $\Phi(W'') = \Phi(T)\hat{\Phi}(A \cap B)\hat{\Phi}([H] \setminus (A \cup B))\Phi(T')$ . We will show using 1-stability of the semiring that  $\Phi(W') + \Phi(W'') + \Phi(W) = \Phi(W') + \Phi(W'')$ , implying  $\Phi(W) + A_{i,j}^{(k)} = A_{i,j}^{(k)}$  as desired.

Thus, it suffices to show  $\Phi(W') + \Phi(W'') + \Phi(W) = \Phi(W') + \Phi(W'')$ . To see this:

$$\begin{aligned}
 & \Phi(W') + \Phi(W'') + \Phi(W) \\
 &= \Phi(T)\hat{\Phi}(A \cap B)\hat{\Phi}(B \setminus A)\hat{\Phi}([H] \setminus (A \cup B))\Phi(T') + \Phi(T)\hat{\Phi}(A \cap B)\hat{\Phi}([H] \setminus (A \cup B))\Phi(T') \\
 &\quad + \Phi(T)\hat{\Phi}(A \cap B)(\hat{\Phi}(B \setminus A))^2\hat{\Phi}([H] \setminus (A \cup B))\Phi(T') \\
 &= \Phi(T)\hat{\Phi}(A \cap B)\Phi([H] \setminus (A \cup B))\Phi(T')(1 + \Phi(B \setminus A) + \Phi(B \setminus A)^2) \\
 &\quad \text{[associative and 1 is the multiplicative identity]} \\
 &= \Phi(T)\hat{\Phi}(A \cap B)\Phi([H] \setminus (A \cup B))\Phi(T')(1 + \Phi(B \setminus A)) \quad \text{[1-stable]} \\
 &= \Phi(W') + \Phi(W'') \quad \blacktriangleleft
 \end{aligned}$$

# Enumeration and Updates for Conjunctive Linear Algebra Queries Through Expressibility


Thomas Muñoz Serrano ✉ 

UHasselt, Data Science Institute, Diepenbeek, Belgium

Cristian Riveros ✉ 

Pontificia Universidad Católica de Chile, Santiago, Chile

Millennium Institute for Foundational Research on Data, Santiago, Chile

Stijn Vansummeren ✉ 

UHasselt, Data Science Institute, Diepenbeek, Belgium

---

## Abstract

Due to the importance of linear algebra and matrix operations in data analytics, there is significant interest in using relational query optimization and processing techniques for evaluating (sparse) linear algebra programs. In particular, in recent years close connections have been established between linear algebra programs and relational algebra that allow transferring optimization techniques of the latter to the former. In this paper, we ask ourselves which linear algebra programs in MATLANG correspond to the free-connex and q-hierarchical fragments of conjunctive first-order logic. Both fragments have desirable query processing properties: free-connex conjunctive queries support constant-delay enumeration after a linear-time preprocessing phase, and q-hierarchical conjunctive queries further allow constant-time updates. By characterizing the corresponding fragments of MATLANG, we hence identify the fragments of linear algebra programs that one can evaluate with constant-delay enumeration after linear-time preprocessing and with constant-time updates. To derive our results, we improve and generalize previous correspondences between MATLANG and relational algebra evaluated over semiring-annotated relations. In addition, we identify properties on semirings that allow to generalize the complexity bounds for free-connex and q-hierarchical conjunctive queries from Boolean annotations to general semirings.

**2012 ACM Subject Classification** Theory of computation → Database theory

**Keywords and phrases** Query evaluation, conjunctive queries, linear algebra, enumeration algorithms

**Digital Object Identifier** 10.4230/LIPIcs.ICDT.2024.12

**Related Version** *Full Paper Version with Proofs*: <https://arxiv.org/abs/2310.04118> [38]

**Funding** Cristian Riveros was funded by ANID – Millennium Science Initiative Program – Code ICM17\_0 and ANID Fondecyt Regular project 1230935. Thomas Muñoz and Stijn Vansummeren were supported by the Bijzonder Onderzoeksfonds (BOF) of Hasselt University (Belgium) under Grants No. BOF21OWB13 and BOF20ZAP02 as well as by the Research Foundation Flanders (FWO) under research project Grant No. G019222N.

## 1 Introduction

Linear algebra forms the backbone of modern data analytics, as most machine learning algorithms are coded as sequences of matrix operations [1, 4, 19, 20, 42]. In practice, linear algebra programs operate over matrices with millions of entries. Therefore, efficient evaluation of linear algebra programs is a relevant challenge for data management systems which has attracted research attention with several proposals in the area [30, 33, 34, 37, 41].

To optimize and evaluate linear algebra programs, we must first agree on the language in which such programs are expressed. There has been a renewed interest in recent years for designing query languages for specifying linear algebra programs and for understanding their



© Thomas Muñoz Serrano, Cristian Riveros, and Stijn Vansummeren;  
licensed under Creative Commons License CC-BY 4.0

27th International Conference on Database Theory (ICDT 2024).

Editors: Graham Cormode and Michael Shekelyan; Article No. 12; pp. 12:1–12:20

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



expressive power [6, 12, 13, 23, 40]. One such proposal is MATLANG [12], a formal matrix query language that consists of only the basic linear algebra operations and whose extensions (e.g., for-MATLANG) achieve the expressive power of most linear algebra operations [23]. Although MATLANG is a theoretical query language, it includes the core of any linear algebra program and, thus, the optimization and efficient evaluation of MATLANG could have a crucial impact on today’s machine learning systems.

In this work, we study the efficient evaluation of MATLANG programs over sparse matrices whose entries are taken from a general semiring. We consider MATLANG evaluation in both the static and dynamic setting. For static evaluation, we want to identify the fragment that one can evaluate by preprocessing the input in linear time to build a data structure for enumerating the output entries with constant-delay. For dynamic evaluation, we assume that matrix entries are updated regularly and we want to maintain the output of a MATLANG query without recomputing it. For this dynamic setting, we aim to identify the MATLANG fragment that one can evaluate by taking linear time in the size of the update to refresh the aforementioned data structure so that it supports constant-delay enumeration of the modified output entries. These guarantees for both scenarios have become the holy grail for algorithmic query processing since, arguably, it is the best that one can achieve complexity-wise in terms of the input, the output, and the cost of an update [5, 8, 11, 16, 17, 31, 35, 36, 39].

To identify the MATLANG fragments with these guarantees, our approach is straightforward but effective. Instead of developing evaluation algorithms from scratch, we establish a direct correspondence between linear algebra and relational algebra to take advantage of the query evaluation results for conjunctive queries. Indeed, prior work has precisely characterized which subfragments of conjunctive queries can be evaluated and updated efficiently [5, 8, 9, 31]. Our main strategy, then, is to link these conjunctive query fragments to corresponding linear algebra fragments. More specifically, our contributions are as follows.

1. We start by understanding the deep connection between positive first-order logic ( $\text{FO}^+$ ) over binary relations and **sum-MATLANG** [23], an extension of MATLANG. We formalize this connection by introducing schema encodings, which specify how relations simulate matrices and vice-versa, forcing a lossless relationship between both. Using this machinery, we show that **sum-MATLANG** and positive first-order logic are equally expressive over any relation, matrix, and matrix dimension (including non-rectangular matrices). Moreover, we show that conjunctive queries (CQ) coincide with **sum-MATLANG** without matrix addition, which we call **conj-MATLANG**. This result forms the basis for linking both settings and translating the algorithmic results from CQ to subfragments of **conj-MATLANG**.
2. We propose **free-connex MATLANG** (**fc-MATLANG**) for static evaluation, a natural MATLANG subfragment that we show to be equally expressive as *free-connex CQ* [5], a subfragment of CQ that allows linear time preprocessing and constant-delay enumeration. To obtain our expressiveness result, we show that free-connex CQs over binary relations are equally expressive as the two-variable fragment of conjunctive  $\text{FO}^+$ , a logical characterization of this class that could be of independent interest.
3. For the dynamic setting we introduce the language **qh-MATLANG**, a MATLANG fragment that we show equally expressive to *q-hierarchical CQ* [9, 31], a fragment of CQ that allows constant update time and constant-delay enumeration.
4. Both free-connex and q-hierarchical CQ are known to characterize the class of CQs that one can evaluate efficiently on Boolean databases. We are interested, however, in evaluating MATLANG queries on matrices featuring entries in a *general semiring*. To obtain the complexity bounds for **fc-MATLANG** and **qh-MATLANG** on general semirings, therefore, we show that the upper and lower bounds for free-connex and q-hierarchical

CQs generalize from Boolean annotations to classes of semirings which includes most semirings used in practice, like the reals. The tight expressiveness connections established in this paper then prove that for such semirings  $\text{fc-MATLANG}$  and  $\text{qh-MATLANG}$  can be evaluated with the same guarantees as their CQ counterparts and that they are optimal: one cannot evaluate any other  $\text{conj-MATLANG}$  query outside this fragment under complexity-theoretic assumptions [9].

An extended version of this paper that includes full proofs of formal statements is available online [38].

**Related work.** In addition to the work that has already been cited above, the following work is relevant. Brijder et al. [13] have shown equivalence between  $\text{MATLANG}$  and  $\text{FO}_3^+$ , the 3-variable fragment of positive first order logic. By contrast, we show equivalence between  $\text{sum-MATLANG}$  and  $\text{FO}^+$ , and study the relationship between the free-connex and q-hierarchical fragments of  $\text{MATLANG}$  and  $\text{FO}^\wedge$ , the conjunctive fragment of positive first order logic.

Geerts et al. [23] previously established a correspondence between  $\text{sum-MATLANG}$  and  $\text{FO}^+$ . However, as we illustrate in [38] their correspondence is (1) restricted to square matrices, (2) asymmetric between the two settings, and (3) encodes matrix instances as databases of more than linear size, making it unsuitable to derive the complexity bounds.

Eldar et al. [18] have recently also generalized complexity bounds for free-connex CQs from Boolean annotations to general semirings. Nevertheless, this generalization is with respect to *direct access*, not enumeration. In their work the focus is to compute aggregate queries, which is achieved by providing direct access to the answers of a query even if the annotated value (aggregation result) is zero. By contrast, in our setting, zero-annotated values must not be reported during the enumeration of query answers. This difference in the treatment of zero leads to a substantial difference in the properties that a semiring must have in order to generalize the existing complexity bounds.

There are deep connections known between the treewidth and the number of variables of a conjunctive  $\text{FO}^+$  formula ( $\text{FO}^\wedge$ ). For example, Kolaitis and Vardi established the equivalence of boolean queries in  $\text{FO}_k^\wedge$ , the  $k$ -variable fragment of  $\text{FO}^\wedge$ , and boolean queries in  $\text{FO}^\wedge$  of treewidth less than  $k$ . Because they focus on boolean queries (i.e., without free variables), this result does not imply our result that for binary queries free-connex  $\text{FO}^\wedge$  equals  $\text{FO}_2^\wedge$ . Similarly, Geerts and Reutter [24] introduce a tensor logic  $\text{TL}$  over binary relations and show that conjunctive expressions in this language that have treewidth  $k$  can be expressed in  $\text{TL}_{k+1}$ , the  $k$ -variable fragment of  $\text{TL}$ . While they do take free variables into account, we show in [38] that there are free-connex conjunctive queries with 2 free variables with treewidth 2 in their formalism – for which their result hence only implies expressibility in  $\text{FO}_3^\wedge$ , not  $\text{FO}_2^\wedge$  as we show here.

Several proposals [30, 33, 34, 37, 41] have been made regarding the efficient evaluation of linear algebra programs in the last few years. All these works focused on query optimization without formal guarantees regarding the preprocessing, updates, or enumeration in query evaluation. To the best of our knowledge, this is the first work on finding subfragments of a linear algebra query language (i.e.,  $\text{MATLANG}$ ) with such efficient guarantees.

## 2 Preliminaries

In this section we recall the main definitions of  $\text{MATLANG}$ , a query language on matrices, and first order logic ( $\text{FO}$ ), a query language on relations.

**Semirings.** We evaluate both languages over arbitrary commutative and non-trivial semirings. A (commutative and non-trivial) *semiring*  $(K, \oplus, \odot, \mathbf{0}, \mathbf{1})$  is an algebraic structure where  $K$  is a non-empty set,  $\oplus$  and  $\odot$  are binary operations over  $K$ , and  $\mathbf{0}, \mathbf{1} \in K$  with  $\mathbf{0} \neq \mathbf{1}$ . Furthermore,  $\oplus$  and  $\odot$  are associative operations,  $\mathbf{0}$  and  $\mathbf{1}$  are the identities of  $\oplus$  and  $\odot$ , respectively,  $\oplus$  and  $\odot$  are commutative operations,  $\odot$  distributes over  $\oplus$ , and  $\mathbf{0}$  annihilates  $K$  (i.e.  $\mathbf{0} \odot k = k \odot \mathbf{0} = \mathbf{0}$ ). We use  $\bigoplus_L$  and  $\bigodot_L$  to denote the  $\oplus$  and  $\odot$  operation over all elements in  $L \subseteq K$ , respectively. Typical examples of semirings are the reals  $(\mathbb{R}, +, \times, 0, 1)$ , the natural numbers  $(\mathbb{N}, +, \times, 0, 1)$ , and the boolean semiring  $\mathbb{B} = (\{\mathbf{t}, \mathbf{f}\}, \vee, \wedge, \mathbf{f}, \mathbf{t})$ .

Henceforth, when we say “semiring” we mean “commutative and non-trivial” semiring. We fix such an arbitrary semiring  $K$  throughout the document. We denote by  $\mathbb{N}_{>0}$  the set of non-zero natural numbers.

**Matrices and size symbols.** A  $K$ -*matrix* (or just matrix) of dimension  $m \times n$  is a  $m \times n$  matrix with elements in  $K$  as its entries. We write  $\mathbf{A}_{ij}$  to denote the  $(i, j)$ -entry of  $\mathbf{A}$ . Matrices of dimension  $m \times 1$  are *column vectors* and those of dimension  $1 \times n$  are *row vectors*. We also refer to matrices of dimension  $1 \times 1$  as *scalars*.

We assume a collection of *size symbols* denoted with greek letters  $\alpha, \beta, \dots$  and assume that the natural number 1 is a valid size symbol. A *type* is a pair  $(\alpha, \beta)$  of size symbols. Intuitively, types represent sets of matrix dimensions. In particular, we obtain dimensions from types by replacing size symbols by elements from  $\mathbb{N}_{>0}$ , where the size symbol 1 is always replaced by the natural number 1. So,  $(\alpha, \beta)$  with  $\alpha \neq 1 \neq \beta$  represents the set of dimensions  $\{(m, n) \mid m, n \in \mathbb{N}_{>0}\}$ , while  $(\alpha, \alpha)$  represents the dimensions  $\{(m, m) \mid m \in \mathbb{N}_{>0}\}$  of square matrices; and  $(\alpha, 1)$  represents the dimensions  $\{(m, 1) \mid m \in \mathbb{N}_{>0}\}$  of column vectors and  $(1, 1)$  represents the dimension  $(1, 1)$  of scalars.

**Schemas and instances.** We assume a set  $\mathcal{M} = \{\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{V}, \dots\}$  of *matrix symbols*, disjoint with the size symbols and denoted by bold uppercase letters. Each matrix symbol  $\mathbf{A}$  has a fixed associated type. We write  $\mathbf{A} : (\alpha, \beta)$  to denote that  $\mathbf{A}$  has type  $(\alpha, \beta)$ .

A matrix *schema*  $\mathcal{S}$  is a finite set of matrix and size symbols. We require that the special size symbol 1 is always in  $\mathcal{S}$ , and that all size symbols occurring in the type of any matrix symbol  $\mathbf{A} \in \mathcal{S}$  are also in  $\mathcal{S}$ . A matrix *instance*  $\mathcal{I}$  over a matrix schema  $\mathcal{S}$  is a function that maps each size symbol  $\alpha$  in  $\mathcal{S}$  to a non-zero natural number  $\alpha^{\mathcal{I}} \in \mathbb{N}_{>0}$ , and maps each matrix symbol  $\mathbf{A} : (\alpha, \beta)$  in  $\mathcal{S}$  to a  $K$ -matrix  $\mathbf{A}^{\mathcal{I}}$  of dimension  $\alpha^{\mathcal{I}} \times \beta^{\mathcal{I}}$ . We assume that for the size symbol 1, we have  $1^{\mathcal{I}} = 1$ , for every instance  $\mathcal{I}$ .

**Sum-Matlang.** Let  $\mathcal{S}$  be a matrix schema. Before defining the syntax of sum-MATLANG, we assume a set  $\mathcal{V} = \{\mathbf{u}, \mathbf{v}, \mathbf{w}, \mathbf{x}, \dots\}$  of *vector variables* over  $\mathcal{S}$ , which is disjoint with matrix and size symbols in  $\mathcal{S}$ . Each such variable  $\mathbf{v}$  has a fixed associated type, which must be a vector type  $(\gamma, 1)$  for some size symbol  $\gamma \in \mathcal{S}$ . We also write  $\mathbf{v} : (\gamma, 1)$  in that case.

The syntax of sum-MATLANG expressions [23] over  $\mathcal{S}$  is defined by the following grammar:

$e ::=$	$\mathbf{A} \in \mathcal{S}$	(matrix symbol)		$\mathbf{v} \in \mathcal{V}$	(vector variable)	
		$e^T$	(transpose)		$e_1 \cdot e_2$	(matrix multiplication)
		$e_1 + e_2$	(matrix addition)		$e_1 \times e_2$	(scalar multiplication)
		$e_1 \odot e_2$	(pointwise multiplication)		$\Sigma \mathbf{v}.e$	(sum-iteration).

In addition, we require that expressions  $e$  are *well-typed*, in the sense that its *type*  $\text{type}(e)$  is correctly defined as follows:

$$\begin{aligned}
\text{type}(\mathbf{A}) &:= (\alpha, \beta) \text{ for a matrix symbol } \mathbf{A}: (\alpha, \beta) \\
\text{type}(\mathbf{v}) &:= (\gamma, 1) \text{ for vector variable } \mathbf{v}: (\gamma, 1) \\
\text{type}(e^T) &:= (\beta, \alpha) \text{ if } \text{type}(e) = (\alpha, \beta) \\
\text{type}(e_1 \cdot e_2) &:= (\alpha, \gamma) \text{ if } \text{type}(e_1) = (\alpha, \beta) \text{ and } \text{type}(e_2) = (\beta, \gamma) \\
\text{type}(e_1 + e_2) &:= (\alpha, \beta) \text{ if } \text{type}(e_1) = \text{type}(e_2) = (\alpha, \beta) \\
\text{type}(e_1 \times e_2) &:= (\alpha, \beta) \text{ if } \text{type}(e_1) = (1, 1) \text{ and } \text{type}(e_2) = (\alpha, \beta) \\
\text{type}(e_1 \odot e_2) &:= (\alpha, \beta) \text{ if } \text{type}(e_1) = \text{type}(e_2) = (\alpha, \beta) \\
\text{type}(\Sigma \mathbf{v}.e) &:= (\alpha, \beta) \text{ if } e: (\alpha, \beta) \text{ and } \text{type}(\mathbf{v}) = (\gamma, 1).
\end{aligned}$$

In what follows, we always consider well-typed expressions and write  $e: (\alpha, \beta)$  to denote that  $e$  is well typed, and its type is  $(\alpha, \beta)$ .

For an expression  $e$ , we say that a vector variable  $\mathbf{v}$  is *bound* if it is under a sum-iteration  $\Sigma \mathbf{v}$ , and *free* otherwise. To evaluate expressions with free vector variables, we require the following notion of a valuation. Fix a matrix instance  $\mathcal{I}$  over  $\mathcal{S}$ . A *vector valuation* over  $\mathcal{I}$  is a function  $\mu$  that maps each vector symbol  $\mathbf{v}: (\gamma, 1)$  to a column vector of dimension  $\gamma^{\mathcal{I}} \times 1$ . Further, if  $\mathbf{b}$  is a vector of dimension  $\gamma^{\mathcal{I}} \times 1$ , then let  $\mu[\mathbf{v} := \mathbf{b}]$  denote the *extended* vector valuation over  $\mathcal{I}$  that coincides with  $\mu$ , except that  $\mathbf{v}: (\gamma, 1)$  is mapped to  $\mathbf{b}$ .

Let  $e: (\alpha, \beta)$  be a sum-MATLANG expression over  $\mathcal{S}$ . When one evaluates  $e$  over a matrix instance  $\mathcal{I}$  and a matrix valuation  $\mu$  over  $\mathcal{I}$ , it produces a matrix  $\llbracket e \rrbracket(\mathcal{I}, \mu)$  of dimension  $\alpha^{\mathcal{I}} \times \beta^{\mathcal{I}}$  such that each entry  $i, j$  satisfies:

$$\begin{aligned}
\llbracket \mathbf{A} \rrbracket(\mathcal{I}, \mu)_{ij} &:= \mathbf{A}_{ij}^{\mathcal{I}} \text{ for } \mathbf{A} \in \mathcal{S} \\
\llbracket \mathbf{v} \rrbracket(\mathcal{I}, \mu)_{ij} &:= \mu(\mathbf{v})_{ij} \text{ for } \mathbf{v} \in \mathcal{V} \\
\llbracket e^T \rrbracket(\mathcal{I}, \mu)_{ij} &:= \llbracket e \rrbracket(\mathcal{I}, \mu)_{ji} \\
\llbracket e_1 + e_2 \rrbracket(\mathcal{I}, \mu)_{ij} &:= \llbracket e_1 \rrbracket(\mathcal{I}, \mu)_{ij} \oplus \llbracket e_2 \rrbracket(\mathcal{I}, \mu)_{ij} \\
\llbracket e_1 \odot e_2 \rrbracket(\mathcal{I}, \mu)_{ij} &:= \llbracket e_1 \rrbracket(\mathcal{I}, \mu)_{ij} \odot \llbracket e_2 \rrbracket(\mathcal{I}, \mu)_{ij} \\
\llbracket \Sigma \mathbf{v}.e \rrbracket(\mathcal{I}, \mu)_{ij} &:= \bigoplus_{k=1}^{\gamma^{\mathcal{I}}} \llbracket e \rrbracket(\mathcal{I}, \mu[\mathbf{v} := \mathbf{b}_k^{\gamma^{\mathcal{I}}}] )_{ij} \\
\llbracket e_1 \cdot e_2 \rrbracket(\mathcal{I}, \mu)_{ij} &:= \bigoplus_k \llbracket e_1 \rrbracket(\mathcal{I}, \mu)_{ik} \odot \llbracket e_2 \rrbracket(\mathcal{I}, \mu)_{kj} \\
\llbracket e_1 \times e_2 \rrbracket(\mathcal{I}, \mu)_{ij} &:= a \odot \llbracket e_2 \rrbracket(\mathcal{I}, \mu)_{ij} \text{ with } \llbracket e_1 \rrbracket(\mathcal{I}, \mu) = [a]
\end{aligned}$$

where  $\mathbf{v}: (\gamma, 1)$  and  $\mathbf{b}_1^n, \mathbf{b}_2^n, \dots, \mathbf{b}_n^n$  are the  $n$ -dimension canonical vectors, namely, the vectors  $[\mathbf{1} \mathbf{0} \dots \mathbf{0}]^T, [\mathbf{0} \mathbf{1} \dots \mathbf{0}]^T, \dots, [\mathbf{0} \mathbf{0} \dots \mathbf{1}]^T$ , respectively.

► **Example 1.** Let  $\mathcal{S} = \{\mathbf{A}\}$  where  $\mathbf{A}: (\alpha, \alpha)$ . Let  $\mathcal{I}$  be an instance over  $\mathcal{S}$  such that  $\alpha^{\mathcal{I}} = 3$  and  $\mathbf{A}^{\mathcal{I}} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$ . Let  $\mathbf{v} \in \mathcal{V}$  where  $\mathbf{v}: (\alpha, 1)$ . The expression  $\Sigma \mathbf{v}.\mathbf{A} \cdot \mathbf{v}$  is well-typed and

$$\llbracket \Sigma \mathbf{v}.\mathbf{A} \cdot \mathbf{v} \rrbracket(\mathcal{I}, \emptyset) = \mathbf{A} \cdot \mathbf{b}_1^3 + \mathbf{A} \cdot \mathbf{b}_2^3 + \mathbf{A} \cdot \mathbf{b}_3^3 = \begin{bmatrix} a_{11} \\ a_{21} \\ a_{31} \end{bmatrix} + \begin{bmatrix} a_{12} \\ a_{22} \\ a_{32} \end{bmatrix} + \begin{bmatrix} a_{13} \\ a_{23} \\ a_{33} \end{bmatrix}.$$

► **Example 2.** Let  $\mathcal{S}$  and  $\mathcal{I}$  be as in Example 1. Let  $\mathbf{v} \in \mathcal{V}$  where  $\mathbf{v}: (\gamma, 1)$ . The expression  $\Sigma \mathbf{v}.\mathbf{A}$  is well-typed and

$$\llbracket \Sigma \mathbf{v}.\mathbf{A} \rrbracket(\mathcal{I}, \emptyset) = \underbrace{\mathbf{A} + \dots + \mathbf{A}}_{\gamma^{\mathcal{I}} \text{ times}}.$$

## 12:6 Enumeration and Updates for Conjunctive Linear Algebra Queries

**Matlang.** MATLANG is a linear algebra query language that is a fragment of sum-MATLANG. Specifically, define the syntax of MATLANG expressions over  $\mathcal{S}$  by the following grammar:

$$e ::= \mathbf{A} \in \mathcal{S} \mid e^T \mid e_1 \cdot e_2 \mid e_1 + e_2 \mid e_1 \times e_2 \mid e_1 \odot e_2 \mid \mathbf{1}^\alpha \mid \mathbf{I}^\alpha$$

for every size symbol  $\alpha \in \mathcal{S}$ . Here,  $\mathbf{1}^\alpha$  and  $\mathbf{I}^\alpha$  denote the *ones-vector* and *identity-matrix*, respectively, of type  $\text{type}(\mathbf{1}^\alpha) = (\alpha, 1)$  and  $\text{type}(\mathbf{I}^\alpha) = (\alpha, \alpha)$ . Their semantics can be defined by using sum-MATLANG as:

$$\llbracket \mathbf{1}^\alpha \rrbracket(\mathcal{I}, \mu) := \llbracket \Sigma \mathbf{v} \cdot \mathbf{v} \rrbracket(\mathcal{I}, \mu) \quad \llbracket \mathbf{I}^\alpha \rrbracket(\mathcal{I}, \mu) := \llbracket \Sigma \mathbf{v} \cdot \mathbf{v} \cdot \mathbf{v}^T \rrbracket(\mathcal{I}, \mu)$$

Note that the original MATLANG version introduced in [12] included an operator for the *diagonalization of a vector*. This operator can be simulated by using the  $\mathbf{I}^\alpha$ -operator and vice versa. Furthermore, we have included the pointwise multiplication  $\odot$  in MATLANG, also known as the *Hadamard product*. This operation will be essential for our characterization results. In [12, 23], the syntax of MATLANG was more generally parameterized by a family of  $n$ -ary functions that could be pointwise applied. Similarly to [13, 22] we do not include such functions here, but leave their detailed study to future work.

**Sum-Matlang queries.** A sum-MATLANG *query*  $\mathbf{Q}$  over a matrix schema  $\mathcal{S}$  is an expression of the form  $\mathbf{H} := e$  where  $e$  is a well-typed sum-MATLANG expression without free vector variables,  $\mathbf{H}$  is a “fresh” matrix symbol that does not occur in  $\mathcal{S}$ , and  $\text{type}(e) = \text{type}(\mathbf{H})$ . When evaluated on a matrix instance  $\mathcal{I}$  over schema  $\mathcal{S}$ ,  $\mathbf{Q}$  returns a matrix instance  $\mathcal{E}$  over the extended schema  $\mathcal{S} \cup \{\mathbf{H}\}$ :  $\mathcal{E}$  coincides with  $\mathcal{I}$  for every matrix and size symbol in  $\mathcal{S}$  and additionally maps  $\mathbf{H}^\mathcal{E} = \llbracket e \rrbracket(\mathcal{I}, \emptyset)$  with  $\emptyset$  denoting the empty vector valuation. We denote the instance resulting from evaluating  $\mathbf{Q}$  by  $\llbracket \mathbf{Q} \rrbracket(\mathcal{I})$ . If  $\mathcal{S}$  is a matrix schema and  $\mathbf{Q}$  a sum-MATLANG query over  $\mathcal{S}$  then we use  $\mathcal{S}(\mathbf{Q})$  to denote the extended schema  $\mathcal{S} \cup \{\mathbf{H}\}$ .

**$K$ -relations.** A  $K$ -relation over a domain of data values  $\mathbb{D}$  is a function  $f: \mathbb{D}^a \rightarrow K$  such that  $f(\vec{d}) \neq 0$  for finitely many  $\vec{d} \in \mathbb{D}^a$ . Here, “ $a$ ” is the *arity* of  $R$ . Since we want to compare relational queries with sum-MATLANG queries, we will restrict our attention in what follows to  $K$ -relations where the domain  $\mathbb{D}$  of data values is the set  $\mathbb{N}_{>0}$ . In this context, we may naturally view a  $K$ -matrix of dimensions  $n \times m$  as a  $K$ -relation such that the entry  $(i, j)$  of the matrix is encoded by the  $K$ -value of the tuple  $(i, j)$  in the relation (see also Section 3).

**Vocabularies and databases.** We assume an infinite set of *relation symbols* together with an infinite and disjoint set of *constant symbols*. Every relation symbol  $R$  is associated with a number, its *arity*, which we denote by  $\text{ar}(R) \in \mathbb{N}$ . A *vocabulary*  $\sigma$  is a finite set of relation and constant symbols. A *database* over  $\sigma$  is a function  $db$  that maps every constant symbol  $c \in \sigma$  to a value  $c^{db}$  in  $\mathbb{N}_{>0}$ ; and every relation symbol  $R \in \sigma$  to a  $K$ -relation  $R^{db}$  of arity  $\text{ar}(R)$ .

**Positive first order logic.** As our relational query language, we will work with the positive fragment of first order logic ( $\text{FO}^+$ ). In contrast to the standard setting in database theory, where the only atomic formulas are relational atoms of the form  $R(\vec{x})$ , we also allow the ability to compare variables with constant symbols. To this end, the following definitions are in order. We assume an infinite set of variables, which we usually denote by  $x, y, z$ . We denote tuples of variables by  $\vec{x}, \vec{y}$ , and so on. A *relational atom* is expression of the form  $R(x_1, \dots, x_k)$  with  $R$  a relation symbol of arity  $k$ . A *comparison atom* is of the form  $x \leq c$  with  $x$  a variable and  $c$  a constant symbol. A *positive first order logic formula* ( $\text{FO}^+$  formula) over a vocabulary  $\sigma$  is an expression generated by the following grammar:

$$\varphi ::= R(\bar{x}) \mid x \leq c \mid \exists \bar{y}. \varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi$$

where  $R$  and  $c$  range over relations and constants in  $\sigma$ , respectively. We restrict ourselves to *safe* formulas, in the sense that in a disjunction  $\varphi_1 \vee \varphi_2$ , we require that  $\varphi_1$  and  $\varphi_2$  have the same set of free variables [2]. The notions of *free and bound variables* are defined as usual in FO. We denote the set of free variables of  $\varphi$  by  $\text{free}(\varphi)$  and its multiset of atoms by  $\text{at}(\varphi)$ .

We evaluate formulas over  $K$ -relations as follows using the well-known semiring semantics [26]. A *valuation* over a set of variables  $X$  is a function  $\nu: X \rightarrow \mathbb{N}_{>0}$  that assigns a value in  $\mathbb{N}_{>0}$  to each variable in  $X$  (recall that  $\mathbb{D} = \mathbb{N}_{>0}$ ). We denote by  $\nu: X$  that  $\nu$  is a valuation on  $X$  and by  $\nu|_Y$  the restriction of  $\nu$  to  $X \cap Y$ . As usual, valuations are extended point-wise to tuples of variables, i.e.,  $\nu(x_1, \dots, x_n) = (\nu(x_1), \dots, \nu(x_n))$ . Let  $\varphi$  be an  $\text{FO}^+$  formula over vocabulary  $\sigma$ . When evaluated on a database  $db$  over vocabulary  $\sigma$ , it defines a mapping  $\llbracket \varphi \rrbracket_{db}$  from valuations over  $\text{free}(\varphi)$  to  $K$  inductively defined as:

$$\begin{aligned} \llbracket R(x_1, \dots, x_k) \rrbracket_{db}(\nu) &:= R^{db}(\nu(x_1), \dots, \nu(x_k)) \\ \llbracket x \leq c \rrbracket_{db}(\nu) &:= \begin{cases} \mathbb{1} & \text{if } \nu(x) \leq c^{db} \\ \mathbb{0} & \text{otherwise} \end{cases} \\ \llbracket \varphi_1 \wedge \varphi_2 \rrbracket_{db}(\nu) &:= \llbracket \varphi_1 \rrbracket_{db}(\nu|_{\text{free}(\varphi_1)}) \odot \llbracket \varphi_2 \rrbracket_{db}(\nu|_{\text{free}(\varphi_2)}) \\ \llbracket \varphi_1 \vee \varphi_2 \rrbracket_{db}(\nu) &:= \llbracket \varphi_1 \rrbracket_{db}(\nu) \oplus \llbracket \varphi_2 \rrbracket_{db}(\nu) \\ \llbracket \exists y. \varphi \rrbracket_{db}(\nu) &:= \bigoplus_{\mu: \text{free}(\varphi) \text{ s.t. } \mu|_{\text{free}(\varphi) \setminus \{y\}} = \nu} \llbracket \varphi \rrbracket_{db}(\mu) \end{aligned}$$

**FO queries.** An  $\text{FO}^+$  query  $Q$  over vocabulary  $\sigma$  is an expression of the form  $H(\bar{x}) \leftarrow \varphi$  where  $\varphi$  is an  $\text{FO}^+$  formula over  $\sigma$ ,  $\bar{x} = (x_1, \dots, x_k)$  is a sequence of (not necessarily distinct) free variables of  $\varphi$ , such that every free variable of  $\varphi$  occurs in  $\bar{x}$ , and  $H$  is a “fresh” relation symbol not in  $\sigma$  with  $\text{ar}(H) = k$ . The formula  $\varphi$  is called the *body* of  $Q$ , and  $H(\bar{x})$  its *head*.

When evaluated over a database  $db$  over  $\sigma$ ,  $Q$  returns a database  $\llbracket Q \rrbracket_{db}$  over the extended vocabulary  $\sigma \cup \{H\}$ . This database  $\llbracket Q \rrbracket_{db}$  coincides with  $db$  for every relation and constant symbol in  $\sigma$ , and maps the relation symbol  $H$  to the  $K$ -relation of arity  $k$  defined as follows. For a sequence of domain values  $\bar{d} = (d_1, \dots, d_k)$ , we write  $\bar{d} \models \bar{x}$  if, for all  $i \neq j$  with  $x_i = x_j$  we also have  $d_i = d_j$ . Clearly, if  $\bar{d} \models \bar{x}$  then the mapping  $\{x_1 \rightarrow d_1, \dots, x_k \rightarrow d_k\}$  is well-defined. Denote this mapping by  $\bar{x} \mapsto \bar{d}$  in this case. Then

$$\llbracket Q \rrbracket_{db}(H) := \bar{d} \mapsto \begin{cases} \llbracket \varphi \rrbracket_{db}(\bar{x} \mapsto \bar{d}) & \text{if } \bar{d} \models \bar{x} \\ \mathbb{0} & \text{otherwise} \end{cases}$$

In what follows, if  $Q$  is a query, then we will often use the notation  $Q(\bar{x})$  to denote that the sequence of the variables in the head of  $Q$  is  $\bar{x}$ . If  $Q$  is a query over  $\sigma$  and  $H(\bar{x})$  its head, then we write  $\sigma(Q)$  for the extended vocabulary  $\sigma \cup \{H\}$ .

We denote by  $\text{FO}^\wedge$  the fragment of  $\text{FO}^+$  formulas in which disjunction is disallowed. A query  $Q = H(\bar{x}) \leftarrow \varphi$  is an  $\text{FO}^\wedge$  query if  $\varphi$  is in  $\text{FO}^\wedge$ . If additionally  $\varphi$  is in prenex normal form, i.e.,  $Q: H(\bar{x}) \leftarrow \exists \bar{y}. a_1 \wedge \dots \wedge a_n$  with  $a_1, \dots, a_n$  (relational or comparison) atoms, then  $Q$  is a *conjunctive query* (CQ). Note that, while the classes of conjunctive queries and  $\text{FO}^\wedge$  queries are equally expressive, for our purposes conjunctive queries are hence formally a syntactic fragment of  $\text{FO}^\wedge$  queries.

An  $\text{FO}^+$  query is *binary* if every relational atom occurring in it (body and head) has arity at most two. Because in sum-MATLANG both the input and output are matrices, our correspondences between sum-MATLANG and  $\text{FO}^+$  will focus on binary queries.

**Discussion.** We have added comparison atoms to  $\text{FO}^+$  in order to establish its correspondence with  $\text{sum-MATLANG}$ . To illustrate why we will need comparison atoms, consider the  $\text{sum-MATLANG}$  expression  $\mathbf{I}^\alpha$  of type  $(\alpha, \alpha)$  that computes the identity matrix. This can be expressed by means of the following CQ  $Q : I(x, x) \leftarrow x \leq \alpha$ . We hence use comparison atoms to align the dimension of the matrices with the domain size of relations.

To make the correspondence hold, we note that in  $\text{MATLANG}$  there is a special size symbol,  $1$ , which is always interpreted as the constant  $1 \in \mathbb{N}$ . This size symbol is used in particular to represent column and row vectors, which have type  $(\alpha, 1)$  and  $(1, \alpha)$  respectively. We endow CQs with the same property in the sense that we will assume in what follows that  $1$  is a valid constant symbol and that  $1^{db} = 1$  for every database  $db$ .

### 3 From matrices to relations and back

Geerts et al. [23] previously established a correspondence between  $\text{sum-MATLANG}$  and  $\text{FO}^+$ . However, as we illustrate in the full version [38] their correspondence is (1) for square matrices, (2) asymmetric, and (3) encodes matrix instances as databases of more than linear size, making it unsuitable to derive the complexity bounds that we are interested in here. In this section, we revisit and generalize the connection between  $\text{sum-MATLANG}$  and  $\text{FO}^+$  by providing translations between the two query languages that works for any matrix schema, are symmetric, and ensure that matrices are encoded as databases of linear size. Towards this goal, we introduce next all the formal machinery to link both settings. We start by determining precisely in what sense relations can encode matrices, or matrices can represent relations, and how this correspondence transfers to queries. Then we show how to generalize the expressibility results in [23] for any matrix sizes and every encoding between schemas.

**How we relate objects.** Let  $\mathbf{A}$  be a matrix of dimension  $m \times n$ . There exist multiple natural ways to encode  $\mathbf{A}$  as a relation, depending on the dimension  $m \times n$ .

- We can always encode  $\mathbf{A}$ , whatever the values of  $m$  and  $n$ , as the binary  $K$ -relation  $R$  such that (1)  $\mathbf{A}_{i,j} = R(i, j)$  for every  $i \leq m, j \leq n$  and (2)  $R(i, j) = \mathbb{0}$  if  $i > m$  or  $j > n$ .
  - If  $\mathbf{A}$  is a column vector ( $n = 1$ ) then we can also encode it as the unary  $K$ -relation  $R$  such that  $\mathbf{A}_{i,1} = R(i)$  for every  $i \leq m$  and  $R(i) = \mathbb{0}$  if  $i > m$ .
  - Similarly, if  $\mathbf{A}$  is a row vector ( $m = 1$ ) then we can encode it as the unary  $K$ -relation  $R$  with  $\mathbf{A}_{1,j} = R(j)$  for every  $j \leq n$  and  $R(i) = \mathbb{0}$  if  $j > n$ .
  - If  $\mathbf{A}$  is a scalar ( $m = n = 1$ ), we can encode it as a nullary  $K$ -relation  $R$  with  $\mathbf{A}_{1,1} = R()$ .
- Note that if  $\mathbf{A}$  is scalar then we can hence encode it by means of a binary relation, a unary relation, or a nullary relation; and if it is a vector we can encode it by a binary or unary relation. In what follows, we write  $\mathbf{A} \simeq R$  to denote that  $R$  encodes  $\mathbf{A}$ .

Conversely, given a (nullary, unary, or binary)  $K$ -relation  $R$  we may interpret this as a matrix of appropriate dimension. Specifically, we say that relation  $R$  is *consistent* with dimension  $m \times n$  if there exists a matrix  $\mathbf{A}$  of dimension  $m \times n$  such that  $\mathbf{A} \simeq R$ . This is equivalent to requiring that relation is  $\mathbb{0}$  on entries outside of  $m \times n$ . Note that, given  $R$  that is consistent with  $m \times n$  there is exactly one matrix  $\mathbf{A} : m \times n$  such that  $\mathbf{A} \simeq R$ .

**How we relate schemas.** A *matrix-to-relational schema encoding* from a matrix schema  $\mathcal{S}$  to a relational vocabulary  $\sigma$  is a function  $\text{Rel} : \mathcal{S} \rightarrow \sigma$  that maps every matrix symbol  $\mathbf{A}$  in  $\mathcal{S}$  to a unary or binary relation symbol  $\text{Rel}(\mathbf{A})$  in  $\sigma$ , and every size symbol  $\alpha$  in  $\mathcal{S}$  to a constant symbol  $\text{Rel}(\alpha)$  in  $\sigma$ . Here,  $\text{Rel}(\mathbf{A})$  can be unary only if  $\mathbf{A}$  is of vector type, and nullary only if  $\mathbf{A}$  is of scalar type. Intuitively,  $\text{Rel}$  specifies which relation symbols will be used to store



the encodings of which matrix symbols. In addition, we require that  $Rel(1) = 1$  and that  $Rel$  is a bijection between  $\mathcal{S}$  and  $\sigma$ . This makes sure that we can always invert  $Rel$ . In what follows, we will only specify that  $Rel$  is a matrix-to-relational schema encoding *on*  $\mathcal{S}$ , leaving the vocabulary  $\sigma$  unspecified. In that case, we write  $Rel(\mathcal{S})$  for the relational vocabulary  $\sigma$ .

Conversely, we define a *relational-to-matrix schema encoding* from  $\sigma$  into  $\mathcal{S}$  as a function  $Mat: \sigma \rightarrow \mathcal{S}$  that maps every relation symbol  $R$  to a matrix symbol  $Mat(R)$  and every constant symbol  $c$  to a size symbol  $Mat(c)$ . We require that all unary relations are mapped to matrix symbols of vector type, either row or column, and that all nullary relations are mapped to matrix symbols of scalar type. Furthermore,  $Mat$  must map  $1 \mapsto 1$  and be bijective. Similarly, we denote by  $Mat(\sigma)$  the matrix schema  $\mathcal{S}$  mapped by  $Mat$ .

Note that the bijection assumption over  $Mat$  imposes some requirements over  $\sigma$  to encode it as matrices. For example,  $Mat$  requires the existence of at least one constant symbol in  $\sigma$  for encoding matrices dimensions, since every matrix symbol has at least one size symbol in its type, and that size symbol is by definition in  $\mathcal{S}$ . The bijection between constant and size symbols is necessary in order to have lossless encoding between both settings.

Given that  $Rel$  and  $Mat$  are bijections between  $\mathcal{S}$  and  $\sigma$ , their inverses  $Rel^{-1}$  and  $Mat^{-1}$  are well defined. Furthermore, by definition we have that  $Rel^{-1}$  and  $Mat^{-1}$  are relational-to-matrix and matrix-to-relational schema encodings, respectively.

**How we relate instances.** We start by specifying how to encode matrix instances as database instances. Fix a matrix-to-relational schema encoding  $Rel$  between  $\mathcal{S}$  and  $Rel(\mathcal{S})$ . Let  $\mathcal{I}$  be a matrix instance over  $\mathcal{S}$  and  $db$  a database over  $Rel(\mathcal{S})$ . We say that  $db$  is a *relational encoding of  $\mathcal{I}$  w.r.t.  $Rel$* , denoted by  $\mathcal{I} \simeq_{Rel} db$ , if

- $\mathbf{A}^{\mathcal{I}} \simeq Rel(\mathbf{A})^{db}$  for every matrix symbol  $\mathbf{A}$  in  $\mathcal{S}$ , and
- $Rel(\alpha)^{db} = \alpha^{\mathcal{I}}$  for every size symbol  $\alpha$  in  $\mathcal{S}$ .

Note that, given  $\mathcal{I}$  and  $Rel$ , the relational encoding  $db$  is uniquely defined. As such, we also denote this database by  $Rel(\mathcal{I})$ .

We now focus on interpreting database instances as matrix instances, which is more subtle. Fix a relational-to-matrix schema encoding  $Mat$  from  $\sigma$  to  $Mat(\sigma)$ . We need to first leverage the consistency requirement from relations to databases. Formally, we say that a database  $db$  over  $\sigma$  is *consistent with  $Mat$*  if for every relation symbol  $R$  in  $\sigma$ ,  $R^{db}$  is consistent with dimension  $c^{db} \times d^{db}$  where  $Mat(R): (Mat(c), Mat(d))$ . In other words, a consistent database specifies the value of each dimension, and the relations are themselves consistent with them.

Let  $db$  be a database over  $\sigma$ , consistent with  $Mat$  and let  $\mathcal{I}$  be a matrix instance of  $Mat(\sigma)$ . We say that  $\mathcal{I}$  is a *matrix encoding of  $db$  w.r.t.  $Mat$* , denoted  $db \simeq_{Mat} \mathcal{I}$ , if

- $Mat(R)^{\mathcal{I}} \simeq R^{db}$  for every relation symbol  $R \in \sigma$ ; and
- $c^{db} = Mat(c)^{\mathcal{I}}$  for every constant symbol  $c \in \sigma$ .

Given  $Mat$  and a consistent database  $db$ , the matrix encoding  $\mathcal{I}$  is uniquely defined. As such, we also denote this instance by  $Mat(db)$ .

From the previous definitions, one notes an asymmetry between both directions. Although an encoding always holds from matrices to relations, we require that the relations are consistent with the sizes (i.e., constants) from relations to matrices. Nevertheless, this asymmetry does not impose a problem when we want to go back and forth, as the next result shows.

► **Proposition 3.** *Let  $Rel$  and  $Mat$  be matrix-to-relational and relational-to-matrix schema encodings from  $\mathcal{S}$  to  $\sigma$  and from  $\sigma$  to  $\mathcal{S}$ , respectively, such that  $Mat = Rel^{-1}$ . Then*

- $Rel^{-1}(Rel(\mathcal{S})) = \mathcal{S}$  and  $Mat^{-1}(Mat(\sigma)) = \sigma$ ;
- $Rel(\mathcal{I})$  is consistent with  $Rel^{-1}$ , for every instance  $\mathcal{I}$  over  $\mathcal{S}$ ;
- $Rel^{-1}(Rel(\mathcal{I})) = \mathcal{I}$ , for every instance  $\mathcal{I}$  over  $\mathcal{S}$ ; and
- $Mat^{-1}(Mat(db)) = db$ , for every  $db$  consistent with  $Mat$ .

## 12:10 Enumeration and Updates for Conjunctive Linear Algebra Queries

The previous proposition is a direct consequence of the definitions; however, it shows that the consistency requirement and schema encodings provide a lossless encoding between the relational and matrix settings. This fact is crucial to formalize the expressiveness equivalence between **sum-MATLANG** and  $\text{FO}^+$ , and their subfragments in the following sections.

**From sum-Matlang to positive-FO.** We first aim to simulate every **sum-MATLANG** query with an  $\text{FO}^+$  query w.r.t. some matrix-to-relational schema encoding. This was already proven in [23] for a different but related setting, and only for *Rel* encodings that map matrix symbols with vector types into unary relations. Here we generalize it to arbitrary encodings.

In what follows, fix a matrix schema  $\mathcal{S}$ . Let  $\mathbf{Q}$  be a **sum-MATLANG** query over  $\mathcal{S}$ , and *Rel* be a matrix-to-relational schema encoding on  $\mathcal{S}(\mathbf{Q})$ . We say that  $\text{FO}^+$  query  $Q$  *simulates*  $\mathbf{Q}$  w.r.t. *Rel* if  $\text{Rel}(\llbracket \mathbf{Q} \rrbracket(\mathcal{I})) = \llbracket Q \rrbracket_{\text{Rel}(\mathcal{I})}$  for every matrix instance  $\mathcal{I}$  over  $\mathcal{S}$ . Note that the definition implies that the output matrix symbol of  $\mathbf{Q}$  must be mapped to the output relation symbol of  $Q$  by *Rel*, since *Rel* is a bijection and the condition must hold for every matrix instance. Indeed, it is equivalent to  $\llbracket \mathbf{Q} \rrbracket(\mathcal{I}) = \text{Rel}^{-1}(\llbracket Q \rrbracket_{\text{Rel}(\mathcal{I})})$ , namely, that one can evaluate  $\mathbf{Q}$  by first evaluating  $\llbracket Q \rrbracket_{\text{Rel}(\mathcal{I})}$  and then mapping the results back.

Next, we show that we can simulate every **sum-MATLANG** query in the relational setting.

► **Proposition 4.** *For every sum-MATLANG query  $\mathbf{Q}$  over  $\mathcal{S}$  and every matrix-to-relational schema encoding *Rel* on  $\mathcal{S}(\mathbf{Q})$ , there exists an  $\text{FO}^+$  query  $Q$  that simulates  $\mathbf{Q}$  w.r.t. *Rel*.*

**From positive-FO to sum-Matlang.** We now aim to simulate every  $\text{FO}^+$  query with a **sum-MATLANG** query. Contrary to the previous direction, the expressiveness result here is more subtle and requires more discussion and additional notions.

Fix a vocabulary  $\sigma$ . Let  $Q$  be a  $\text{FO}^+$  query over  $\sigma$  and let *Mat* be relational-to-matrix schema encoding on  $\sigma(Q)$ . We say that a matrix query  $\mathbf{Q}$  *simulates*  $Q$  w.r.t. *Mat* if  $\text{Mat}(\llbracket Q \rrbracket_{db}) = \llbracket \mathbf{Q} \rrbracket(\text{Mat}(db))$  for every database  $db$  consistent with *Mat*. We note again that this definition implies that the input vocabulary and output relation symbol of  $Q$  coincides with the input schema and output matrix symbol of  $\mathbf{Q}$ , respectively. Further, it is equivalent that  $\llbracket Q \rrbracket_{db} = \text{Mat}^{-1}(\llbracket \mathbf{Q} \rrbracket(\text{Mat}(db)))$ .

Before stating how to connect  $\text{FO}^+$  with **sum-MATLANG**, we need to overcome the following problem: a  $\text{FO}^+$  query can use the same variable within different relational atoms, which can be mapped to matrix symbols of different types. For an illustrative example of this problem, consider the query

$$Q: H(x, y) \leftarrow R(x, y), S(y, z)$$

and a relational-to-matrix schema encoding such that *Mat* maps  $R$  and  $S$  to symbols of type  $(\alpha, \beta)$ ,  $H$  to a symbol of type  $(\beta, \beta)$ , and  $c$  and  $d$  to  $\alpha$  and  $\beta$ , respectively. For a consistent database  $db$  w.r.t. *Mat*, we could have that  $R$  and  $S$  are consistent with  $c^{db} \times d^{db}$ , but  $H$  is not consistent with  $d^{db} \times d^{db}$  if  $d^{db} < c^{db}$ . Moreover, *Mat* bounds variable  $y$  with different sizes  $c^{db}$  and  $d^{db}$ . It is then problematic to simulate  $Q$  under *Mat* in **sum-MATLANG** because **sum-MATLANG** expressions need to be well-typed.

Given the previous discussion, the *well-typedness* definition of a  $\text{FO}^+$  formula is necessary. Let *Mat* be a relational-to-matrix schema encoding on  $\sigma$ . Given a  $\text{FO}^+$  formula  $\varphi$  over  $\sigma$  and a function  $\tau$  from  $\text{free}(\varphi)$  to size symbols in  $\text{Mat}(\sigma)$ , define the rule  $\text{Mat} \vdash \varphi: \tau$  inductively as shown in Figure 1, where  $\tau_1 \sim \tau_2$  if and only if  $\tau_1(x) = \tau_2(x)$  for every  $x \in \text{dom}(\tau_1) \cap \text{dom}(\tau_2)$ . We say that  $\varphi$  over  $\sigma$  is *well-typed* w.r.t. *Mat* if there exists such a function  $\tau$  such that  $\text{Mat} \vdash \varphi: \tau$ . Note that if  $\varphi$  is well-typed, then there is a unique  $\tau$  such that  $\text{Mat} \vdash \varphi: \tau$ .

$$\begin{array}{c}
\frac{\text{type}(\text{Mat}(R)) = (\alpha, \beta)}{\text{Mat} \vdash R(x_1, x_2): \{x_1 \mapsto \alpha, x_2 \mapsto \beta\}} \qquad \frac{\text{type}(\text{Mat}(R)) = (\alpha, 1) \text{ or } (1, \alpha)}{\text{Mat} \vdash R(x): \{x \mapsto \alpha\}} \\
\frac{\text{type}(\text{Mat}(R)) = (1, 1)}{\text{Mat} \vdash R(): \{\}} \qquad \frac{}{\text{Mat} \vdash x \leq c: \{x \mapsto \text{Mat}(c)\}} \qquad \frac{\text{Mat} \vdash \varphi: \tau}{\text{Mat} \vdash \exists \bar{y}. \varphi: \tau|_{\text{free}(\varphi) \setminus \bar{y}}} \\
\frac{\text{Mat} \vdash \varphi_1: \tau_1, \text{Mat} \vdash \varphi_2: \tau_2 \text{ and } \tau_1 \sim \tau_2}{\text{Mat} \vdash \varphi_1 \wedge \varphi_2: \tau_1 \cup \tau_2} \qquad \frac{\text{Mat} \vdash \varphi_1: \tau_1, \text{Mat} \vdash \varphi_2: \tau_2 \text{ and } \tau_1 \sim \tau_2}{\text{Mat} \vdash \varphi_1 \vee \varphi_2: \tau_1 \cup \tau_2}
\end{array}$$

■ **Figure 1** Well-typedness of  $\text{FO}^+$  formulas under relational-to-matrix mapping  $\text{Mat}$ .

Now, let  $Q: H(\bar{x}) \leftarrow \varphi$  be a binary  $\text{FO}^+$  query over  $\sigma$  and  $\text{Mat}$  be a matrix encoding specification on  $\sigma(Q)$ . We say that  $Q$  is *well-typed* w.r.t.  $\text{Mat}$  if  $\varphi$  is well-typed w.r.t.  $\text{Mat}$  and for  $\tau$  such that  $\text{Mat} \vdash \varphi: \tau$ , we have:

- if  $\bar{x} = (x_1, x_2)$ , then  $\text{type}(\text{Mat}(H)) = (\tau(x_1), \tau(x_2))$ ; or
- if  $\bar{x} = (x)$ , then  $\text{type}(\text{Mat}(H))$  is either  $(\tau(x), 1)$  or  $(1, \tau(x))$ .

We write  $\text{Mat} \vdash Q: \tau$  to indicate that  $Q$  is well-typed w.r.t.  $\text{Mat}$ , and  $\tau$  is the unique function testifying to well-typedness of  $\text{FO}^+$  formula of  $Q$ . We note that that we can show that the query obtained by Proposition 4 is always well-typed.

The next proposition connects well-typedness with consistency.

► **Proposition 5.** *For binary  $\text{FO}^+$  query  $Q: H(\bar{x}) \leftarrow \varphi$  over a vocabulary  $\sigma$ , if  $\text{Mat} \vdash Q: \tau$  then for any db consistent with  $\text{Mat}$  we have:*

- *If  $\bar{x} = (x_1, x_2)$  then  $\llbracket Q \rrbracket_{db}(H)$  is consistent with dimension  $\tau(x_1)^{db} \times \tau(x_2)^{db}$ .*
- *If  $\bar{x} = (x)$  then  $\llbracket Q \rrbracket_{db}(H)$  is consistent with both dimension  $\tau(x)^{db} \times 1$  and  $1 \times \tau(x)^{db}$ .*

We have now all the formal machinery to state how to simulate every  $\text{FO}^+$  query over relations with a sum-MATLANG query over matrices.

► **Proposition 6.** *For every binary  $\text{FO}^+$  query  $Q$  over a vocabulary  $\sigma$  and every relational-to-matrix schema encoding  $\text{Mat}$  on  $\sigma(Q)$  such that  $Q$  is well typed w.r.t.  $\text{Mat}$  there exists a sum-MATLANG query  $\mathbf{Q}$  that simulates  $Q$  w.r.t.  $\text{Mat}$ .*

**Conjunctive Matlang.** Taking into account the correspondence between sum-MATLANG and  $\text{FO}^+$  established by Propositions 5 and 6, in what follows we say that matrix query language  $\mathcal{L}_M \subseteq \text{sum-MATLANG}$  and relational language  $\mathcal{L}_R \subseteq \text{FO}^+$  are *equivalent* or *equally expressive* if (1) for every matrix query  $\mathbf{Q} \in \mathcal{L}_M$  over a matrix schema  $\mathcal{S}$  and every matrix-to-relational schema encoding  $\text{Rel}$  on  $\mathcal{S}(\mathbf{Q})$  there exists a query  $Q \in \mathcal{L}_R$  that simulates  $\mathbf{Q}$  w.r.t.  $\text{Rel}$  and is well-typed w.r.t.  $\text{Rel}^{-1}$ ; and (2) for every binary query  $Q \in \mathcal{L}_R$  over a vocabulary  $\sigma$  and every relational-to-matrix schema encoding  $\text{Mat}$  such that  $Q$  is well-typed w.r.t.  $\text{Mat}$  there exists  $\mathbf{Q} \in \mathcal{L}_M$  that simulates  $Q$  w.r.t.  $\text{Mat}$ .

Let conj-MATLANG be the sum-MATLANG fragment that includes all operations except matrix addition (+). Then we can derive the following characterization of CQs.

► **Corollary 7.** *conj-MATLANG and conjunctive queries are equally expressive.*

While this result is a consequence of the connection between sum-MATLANG and  $\text{FO}^+$ , it provides the basis to explore the fragments of conj-MATLANG that correspond to fragments of CQ, like free-connex or q-hierarchical CQ. We determine these fragments in the next sections.

#### 4 The fragment of free-connex queries

In this section, we specialize the correspondence of Corollary 7 between conj-MATLANG and CQs to *free-connex* CQs [5]. Free-connex CQs are a subset of acyclic CQs that allow efficient enumeration-based query evaluation: in the Boolean semiring and under data complexity they allow to enumerate the query result  $\llbracket Q \rrbracket_{db}(H)$  of free-connex CQ  $Q: H(\bar{x}) \leftarrow \varphi$  with *constant delay* after a preprocessing phase that is linear in  $db$ . In fact, under complexity-theoretic assumptions, the class of CQs that admits constant delay enumeration after linear time preprocessing is precisely the class of free-connex CQs [5].

**Acyclic and free-connex CQs.** A CQ  $Q: H(\bar{x}) \leftarrow \exists \bar{y}. a_1 \wedge \dots \wedge a_n$  is called *acyclic* [7,10,21] if it has a *join-tree*, i.e. an undirected tree  $T = (V, E)$  with  $V$  the set  $\{a_1, \dots, a_n\}$  of atoms in the body and where for each variable  $z$  occurring in  $Q$  the set  $\{a_i \in V \mid z \in \text{vars}(a_i)\}$  induces a connected subtree of  $T$ . Note that we consider inequality predicates as unary. Furthermore,  $Q$  is *free-connex* [5,11] if it is acyclic and the query  $Q'$  obtained by adding the head atom  $H(\bar{x})$  to the body of  $Q$  is also acyclic. The second condition forbids queries like  $H(x, y) \leftarrow \exists z. A(x, z) \wedge B(z, y)$  since when we adjoin the head to the body we get  $\exists z. A(x, z) \wedge B(z, y) \wedge H(x, y)$ , which is cyclic. We will usually refer to free-connex CQs simply as fc-CQs in what follows.

To identify the sum-MATLANG fragment that corresponds to fc-CQs, we find it convenient to first observe the following correspondence between fc-CQs and  $\text{FO}_2^\wedge$ , the two-variable fragment of  $\text{FO}^\wedge$ . Here, a formula  $\varphi$  in  $\text{FO}^\wedge$  is said to be in  $\text{FO}_2^\wedge$  if the set of all variables used in  $\varphi$  (free or bound) is of cardinality at most two. So,  $\exists y \exists z. A(x, y) \wedge B(y, z)$  is not in  $\text{FO}_2^\wedge$ , but the equivalent formula  $\exists y. (A(x, y) \wedge \exists x. B(y, x))$  is. An  $\text{FO}_2^\wedge$  query is a binary query whose body is an  $\text{FO}_2^\wedge$  formula. Recall that a query is binary if every relational atom occurring in its body and head have arity at most two.

► **Theorem 8.** *Binary free-connex CQs and  $\text{FO}_2^\wedge$  queries are equally expressive.*

We find this a remarkable characterization of the fc-CQs on binary relations that, to the best of our knowledge, it is new. Moreover, this result motivates the fragment of MATLANG that characterizes fc-CQs.

**Free-connex MATLANG.** Define fc-MATLANG to be the class of all MATLANG expressions generated by the grammar:

$$e ::= \mathbf{A} \mid \mathbf{1}^\alpha \mid \mathbf{I}^\alpha \mid e^T \mid e_1 \times e_2 \mid e_1 \odot e_2 \mid e_1 \cdot v_2 \mid v_1 \cdot e_2$$

where  $v_1$  and  $v_2$  are fc-MATLANG expressions with type  $(\alpha, 1)$  or  $(1, \alpha)$ . In other words, matrix multiplication  $e_1 \cdot e_2$  is only allowed when at least one of  $e_1$  or  $e_2$  has a row or column vector type.

Interestingly, we are able to show that  $\text{FO}_2^\wedge$  and fc-MATLANG are equally expressive.

► **Theorem 9.** *fc-MATLANG and  $\text{FO}_2^\wedge$  are equally expressive.*

From Theorem 8 and Theorem 9 we obtain:

► **Corollary 10.** *fc-MATLANG and binary free-connex CQs are equally expressive.*

## 5 The fragment of q-hierarchical queries

We next specialize the correspondence between conj-MATLANG and CQs to *q-hierarchical* CQs [5]. The q-hierarchical CQs form a fragment of the free-connex CQs that, in addition to supporting constant delay enumeration after linear time preprocessing, have the property that every single-tuple update (insertion or deletion) to the input database can be processed in constant time, after which the enumeration of the updated query result can again proceed with constant delay [9].

**Q-hierarchical CQs.** Let  $Q: H(\bar{x}) \leftarrow \exists \bar{y}. a_1 \wedge \dots \wedge a_n$  be a CQ. For every variable  $x$ , define  $at(x)$  to be the set  $\{a_i \mid x \in \text{var}(a_i)\}$  of relational atoms that mention  $x$ . Note that, contrary to acyclic queries, here we make the distinction between relational atoms and inequalities. Then  $Q$  is q-hierarchical if for any two variables  $x, y$  the following is satisfied:

1.  $at(x) \subseteq at(y)$  or  $at(x) \supseteq at(y)$  or  $at(x) \cap at(y) = \emptyset$ , and
2. if  $x \in \bar{x}$  and  $at(x) \subsetneq at(y)$  then  $y \in \bar{x}$ .

For example,  $H(x) \leftarrow \exists y. A(x, y) \wedge U(x)$  is q-hierarchical. By contrast, the variant  $H(x) \leftarrow \exists y. A(x, y) \wedge U(y)$  is not q-hierarchical, as it violates the second condition. Furthermore,  $H(x, y) \leftarrow A(x, y) \wedge U(x) \wedge V(y)$  violates the first condition, and is also not q-hierarchical. Note that all these examples are free-connex. We refer to q-hierarchical CQs simply as qh-CQs.

**Q-hierarchical Matlang.** The fragment of sum-MATLANG that is equivalent to qh-CQs is a two-layered language where expressions in a higher layer can only be built from the lower layer. This lower layer, called **simple-MATLANG**, is a fragment of **fc-MATLANG** defined as:

$$e ::= \mathbf{A} \mid \mathbf{1}^\alpha \mid \mathbf{I}^\alpha \mid e^T \mid e_1 \times e_2 \mid e_1 \odot e_2 \mid e \cdot \mathbf{1}^\alpha.$$

Note that in **simple-MATLANG** matrix multiplication is further restricted to matrix-vector multiplication with the ones vector. Intuitively, all **simple-MATLANG** expressions can already define q-hierarchical CQs like  $H(x) \leftarrow \exists y. A(x, y) \wedge U(x)$ , but it cannot define cross-products like  $H(x, y) \leftarrow A(x) \wedge B(y)$ , which are q-hierarchical. For this reason, we need to enhance **simple-MATLANG** with the higher layer. Specifically, we define **qh-MATLANG** as follows:

$$e ::= e_1 \mid e_1 \odot (e_2 \cdot (\mathbf{1}^\alpha)^T) \mid (\mathbf{1}^\alpha \cdot e_1) \odot e_2 \mid (\mathbf{1}^\alpha \cdot e_1) \odot (e_2 \cdot (\mathbf{1}^\alpha)^T)$$

where  $e_1$  and  $e_2$  are **simple-MATLANG** expressions. Note that the subexpressions  $\mathbf{1}^\alpha \cdot e_1$  and  $e_2 \cdot (\mathbf{1}^\alpha)^T$  are valid if  $e_1$  and  $e_2$  have a row and column vector type, respectively. Then, both subexpressions are useful for expanding vector-type expressions into a matrix-type expression.

The **qh-MATLANG** syntax does not allow expressions like, for example,  $\mathbf{1}^\alpha \cdot e_1$  where  $e_1$  is a **simple-MATLANG** expression. Nevertheless, one can define this expression alternatively as  $(\mathbf{1}^\alpha \cdot e_1) \odot (\mathbf{1}^\alpha \cdot (\mathbf{1}^\alpha)^T)$ . For presentational purposes, we decided to define **qh-MATLANG** as simple as possible, leaving out some expressions in **fc-MATLANG** that are not in **qh-MATLANG**, although an equivalent **qh-MATLANG** expression defines it.

► **Theorem 11.** *qh-MATLANG and binary q-hierarchical CQs are equally expressive.*

## 6 Efficient evaluation of free-connex and q-hierarchical queries

Now that we have precise connections between subfragments of **MATLANG** and subfragments of CQ, we can use these connections to derive efficient evaluation algorithms for **MATLANG**. Unfortunately, to apply the algorithms for CQ, we must first face two problems: (1) the

## 12:14 Enumeration and Updates for Conjunctive Linear Algebra Queries

evaluation algorithms for fc-CQs and qh-CQs are usually restricted to the Boolean semiring, and (2) these algorithms are for CQ without inequalities (comparison atoms). To overcome these problems, we need to revisit the fc-CQs and qh-CQs evaluation problem, generalize the algorithmic results to other semirings (e.g.,  $\mathbb{R}$ ), and extend the results when queries have also inequalities. Only then can we derive efficient algorithms for the fragments of fc-MATLANG and qh-MATLANG.

**The evaluation setting.** In our setting, we consider query evaluation as an enumeration problem. Specifically, let  $Q$  be a CQ over vocabulary  $\sigma$ ,  $H$  its relation symbol in the head, and  $\mathcal{K}$  a semiring. We define the evaluation problem  $\text{Eval}(Q, \sigma, \mathcal{K})$  as follows:

<b>Problem:</b>	$\text{Eval}(Q, \sigma, \mathcal{K})$
<b>Input:</b>	A $\mathcal{K}$ -database $db$ over $\sigma$
<b>Output:</b>	Enumerate $\{(\bar{d}, \llbracket Q \rrbracket_{db}(H)(\bar{d})) \mid \llbracket Q \rrbracket_{db}(H)(\bar{d}) \neq \mathbb{0}\}$ .

In other words, the evaluation problems ask to retrieve the set of all tuples output by  $Q$  on  $db$ , together with their non-zero annotations. Similarly, we can define the evaluation problem  $\text{Eval}(\mathbf{Q}, \mathcal{S}, \mathcal{K})$  for a conj-MATLANG query  $\mathbf{Q}$  over a matrix schema  $\mathcal{S}$  where we aim to enumerate the non-zero entries of  $\llbracket \mathbf{Q} \rrbracket(\mathcal{I})$  given as input a matrix instance  $\mathcal{I}$  over  $\mathcal{S}$ , represented sparsely as the set of its non-zero entries.

As it is standard in the area, we consider enumeration algorithms on the Random Access Machine (RAM) with uniform cost measure [3]. We assume that semiring values can be represented in  $\mathcal{O}(1)$  space and that the semiring operations  $\oplus$  and  $\odot$  can be evaluated in  $\mathcal{O}(1)$  time. We say that  $\text{Eval}(\mathbf{Q}, \mathcal{S}, \mathcal{K})$  can be evaluated with *linear-time preprocessing and constant-delay*, if there exists an enumeration algorithm that takes  $\mathcal{O}(\|db\|)$  time to preprocess the input database  $db$ , and then retrieves each output  $(\bar{d}, \llbracket Q \rrbracket_{db}(H)(\bar{d}))$  one by one, without repetitions, and with constant-delay per output. Here, the size of database  $db$  is defined to be the number of non-zero entries. The same extends to  $\text{Eval}(\mathbf{Q}, \mathcal{S}, \mathcal{K})$  as expected. Note that we measure the time and delay in *data complexity* as is standard in the literature [5, 8, 9, 32].

**Evaluation of free-connex queries.** We are ready to state the algorithmic results for fc-CQ and fc-MATLANG. A semiring  $(K, \oplus, \odot, \mathbb{0}, \mathbb{1})$  is *zero-divisor free* if, for all  $a, b \in K$ ,  $a \odot b = \mathbb{0}$  implies  $a = \mathbb{0}$  or  $b = \mathbb{0}$ . A zero-divisor free semiring is called a *semi-integral domain* [25]. Note that semirings used in practice, like  $\mathbb{B}$ ,  $\mathbb{N}$ , and  $\mathbb{R}$ , are semi-integral domains.

► **Theorem 12.** *Let  $\mathcal{K}$  be a semi-integral domain. For every free-connex query  $Q$  over  $\sigma$ ,  $\text{Eval}(Q, \sigma, \mathcal{K})$  can be evaluated with linear-time preprocessing and constant-delay. In particular,  $\text{Eval}(\mathbf{Q}, \mathcal{S}, \mathcal{K})$  can also be evaluated with linear-time preprocessing and constant-delay for every fc-MATLANG  $\mathbf{Q}$  over  $\mathcal{S}$ .*

The semi-integral condition is necessary to ensure that zero outputs could only be produced by some zero entries. For instance, consider a semiring  $(K, \oplus, \odot, \mathbb{0}, \mathbb{1})$  such that there exist  $a, b \in K$  where  $a \neq \mathbb{0}$ ,  $b \neq \mathbb{0}$  and  $a \odot b = \mathbb{0}$ . Further, consider the query  $Q : H(x, y) \leftarrow R(x) \wedge S(y)$  over the previous semiring. Let  $R$  and  $S$  be relation symbols with arity one and  $db$  a database over  $\sigma = \{R, S\}$  such that  $R(1) = a$ ;  $R(2) = a$ ;  $S(1) = b$  and  $S(2) = b$ . Then, the output of  $\text{Eval}(Q, \sigma, \mathcal{K})$  with input  $db$  is empty, although the body can be instantiated in four different ways, all of them producing  $\mathbb{0}$  values.



We derive the enumeration algorithm for  $\text{Eval}(Q, \sigma, \mathcal{K})$  by extending the algorithms in [5] for any semi-integral domain  $\mathcal{K}$  and taking care of the inequalities in  $Q$ . We derive the enumeration algorithm for  $\text{Eval}(\mathbf{Q}, \mathcal{S}, \mathcal{K})$  by reducing to  $\text{Eval}(Q, \sigma, \mathcal{K})$ , using Corollary 10.

To illustrate the utility of Theorem 12, consider the fc-MATLANG query  $\mathbf{A} \odot (\mathbf{U} \cdot \mathbf{V}^T)$  where  $\mathbf{U}, \mathbf{V}$  are column vectors. When this query is evaluated in a bottom-up fashion, the subexpression  $(\mathbf{U} \cdot \mathbf{V}^T)$  will generate partial results of size  $\|\mathbf{U}\| \|\mathbf{V}\|$ , causing the entire evaluation to be of complexity  $\Omega(\|\mathbf{A}\| + \|\mathbf{U}\| \|\mathbf{V}\|)$ . By contrast, Theorem 12 tells us that we may evaluate the query in time  $\mathcal{O}(\|\mathbf{A}\| + \|\mathbf{U}\| + \|\mathbf{V}\|)$ .

For lower bounds, we can also extend the results in [5] and [8] for the relational setting under standard complexity assumptions. As in previous work, our lower bounds are for CQ *without self-joins* and we need some additional restrictions for inequalities. Specifically, we say that an inequality  $x \leq c$  in  $Q$  is *covered*, if there exists a relational atom in  $Q$  that mention  $x$ . We say that  $Q$  is *constant-disjoint* if (i) for all covered inequalities  $x \leq c$  we have  $c \neq 1$ , and (ii) for all pairs  $(x \leq c, y \leq d)$  in  $Q$  of covered inequality  $x \leq c$  and non-covered inequality  $y \leq d$  if  $c = d$  then  $y \notin \text{free}(Q)$ . In other words, if a constant symbol other than 1 occurs in both a covered and non-covered inequality in  $Q$ , then it occurs with a bound variable in the non-covered inequality.

A semiring  $\mathcal{K} = (K, \oplus, \odot, \mathbf{0}, \mathbf{1})$  is *zero-sum free* [27, 28] if for all  $a, b \in K$  it holds that  $a \oplus b = \mathbf{0}$  implies  $a = b = \mathbf{0}$ . We are ready to extend the lower bound in [5, 8] as follows.

► **Theorem 13.** *Let  $Q$  be a CQ over  $\sigma$  without self-joins and constant-disjoint. Let  $\mathcal{K}$  be a semiring such that the subsemiring generated by  $\mathbf{0}_{\mathcal{K}}$  and  $\mathbf{1}_{\mathcal{K}}$  is zero-sum free. If  $\text{Eval}(Q, \sigma, \mathcal{K})$  can be evaluated with linear-time preprocessing and constant-delay, then  $Q$  is free-connex, unless either the Sparse Boolean Matrix Multiplication, the Triangle Detection, or the  $(k, k + 1)$ -Hyperclique conjecture is false.*

It is important to note that most semirings used in practice, like  $\mathbb{B}$ ,  $\mathbb{N}$ , and  $\mathbb{R}$ , are such that the subsemiring generated by  $\mathbf{0}_{\mathcal{K}}$  and  $\mathbf{1}_{\mathcal{K}}$  is zero-sum free. Furthermore, the Sparse Boolean Matrix Multiplication conjecture, the Triangle Detection conjecture, and the  $(k, k + 1)$ -Hyperclique conjecture are standard complexity assumptions used by previous works [5, 8] (see the formal statements in the full version [38]).

Unfortunately, given the asymmetry between the relational and matrix settings, the lower bounds do not immediately transfer from the relational to the matrix setting. Specifically, we need a syntactical restriction for conj-MATLANG that implies the constant-disjointedness restriction in the translation of Theorem 8. Intuitively, this happens when both dimensions of a conj-MATLANG query  $\mathbf{H} := e$  are fixed by matrix symbols in  $\mathcal{S}$ . For example, the expressions  $\mathbf{A} \odot (\mathbf{U} \cdot \mathbf{V}^T)$  and  $\Sigma \mathbf{v}. \mathbf{A} \cdot \mathbf{v}$  have both dimensions (i.e., row and column) fixed by  $\mathbf{A}$  where  $\mathbf{U}, \mathbf{V}$  are column vectors. Instead, the expression  $\mathbf{U} \cdot (\mathbf{1}^\alpha)^T$  does not, since its column dimension depends on the value assigned for  $\alpha$  and is not necessarily fixed by  $\mathbf{U}$ . Formally,  $\text{FixDim}(e)$  is the set of *fixed dimensions* of a conj-MATLANG expression  $e$  and it is inductively defined as follows:

$$\begin{aligned} \text{FixDim}(\mathbf{A}) &:= \{0, 1\} \\ \text{FixDim}(\mathbf{v}) &:= \{1\} \\ \text{FixDim}(e^T) &:= \{(i + 1) \bmod 2 \mid i \in \text{FixDim}(e)\} \\ \text{FixDim}(e_1 \cdot e_2) &:= (\text{FixDim}(e_1) \setminus \{1\}) \cup (\text{FixDim}(e_2) \setminus \{0\}) \\ \text{FixDim}(e_1 \times e_2) &:= \text{FixDim}(e_2) \\ \text{FixDim}(e_1 \odot e_2) &:= \text{FixDim}(e_1) \cup \text{FixDim}(e_2) \\ \text{FixDim}(\Sigma \mathbf{v}. e) &:= \text{FixDim}(e). \end{aligned}$$

An expression  $e$  has *guarded dimensions* if  $\text{FixDim}(e) = \{0, 1\}$ .



## 12:16 Enumeration and Updates for Conjunctive Linear Algebra Queries

The former is straightforwardly extended to a conj-MATLANG query  $\mathbf{Q}$ , where  $\text{FixDim}(\mathbf{Q})$  stands for  $\text{FixDim}(e)$ . The following lower bound is now attainable.

► **Corollary 14.** *Let  $\mathbf{Q}$  be a conj-MATLANG query over  $\mathcal{S}$  such that  $\mathbf{Q}$  does not repeat matrix symbols and  $\mathbf{Q}$  has guarded dimensions. Let  $\mathcal{K}$  be a semiring such that the subsemiring generated by  $0_{\mathcal{K}}$  and  $1_{\mathcal{K}}$  is zero-sum free. If  $\text{Eval}(\mathbf{Q}, \mathcal{S}, \mathcal{K})$  can be evaluated with linear-time preprocessing and constant-delay, then  $\mathbf{Q}$  is equivalent to a fc-MATLANG query, unless either the Sparse Boolean Matrix Multiplication, the Triangle Detection, or the  $(k, k+1)$ -Hyperclique conjecture is false.*

**The dynamic evaluation setting.** We move now to the dynamic query evaluation both in the relational and matrix scenarios. Specifically, we consider the following set of updates. Recall that  $\mathcal{K} = (K, \oplus, \odot, \mathbf{0}, \mathbf{1})$  is a semiring and  $\sigma$  a vocabulary.

- A single-tuple *insertion* (over  $\mathcal{K}$  and  $\sigma$ ) is an operation  $u = \text{insert}(R, \bar{d}, k)$  with  $R \in \sigma$ ,  $\bar{d}$  a tuple of arity  $\text{ar}(R)$ , and  $k \in K$ . When applied to a database  $db$  it induces the database  $db + u$  that is identical to  $db$ , but  $R^{db+u}(\bar{d}) = R^{db}(\bar{d}) \oplus k$ .
- A single-tuple *deletion* (over  $\mathcal{K}$  and  $\sigma$ ) is an expression  $u = \text{delete}(R, \bar{d})$  with  $R \in \sigma$  and  $\bar{d}$  a tuple of arity  $\text{ar}(R)$ . When applied to a database  $db$  it induces the database  $db + u$  that is identical to  $db$ , but  $R^{db+u}(\bar{d}) = \mathbf{0}$ .

Notice that if every element in  $\mathcal{K}$  has an additive inverse (i.e.,  $\mathcal{K}$  is a ring), one can simulate a deletion with an insertion. However, if this is not the case (e.g.,  $\mathbb{B}$  or  $\mathbb{N}$ ), then a single-tuple deletion is a necessary operation.

We define the dynamic query evaluation problem  $\text{DynEval}(Q, \sigma, \mathcal{K})$  as the extension of  $\text{Eval}(Q, \sigma, \mathcal{K})$  under the set of updates  $U$  of all single-tuple insertions and deletions over  $\mathcal{K}$  and  $\sigma$ . We say that  $\text{DynEval}(Q, \sigma, \mathcal{K})$  can be evaluated dynamically with *constant-time update and constant-delay*, if  $\text{Eval}(Q, \sigma, \mathcal{K})$  can be evaluated with linear-time preprocessing and constant delay, and, moreover, for every update  $u \in U$ , it takes constant-time to update the state of the algorithm from  $db$  to  $db + u$  so that, immediately after, we can retrieve each output  $(\bar{d}, \llbracket Q \rrbracket_{db+u}(H)(\bar{d}))$  one by one, without repetitions, and with constant-delay per output. Similarly, we define the dynamic query evaluation problem  $\text{DynEval}(\mathbf{Q}, \mathcal{S}, \mathcal{K})$  of  $\text{Eval}(\mathbf{Q}, \mathcal{S}, \mathcal{K})$  for the matrix setting, requiring the same dynamic guarantees.

Note that updates allow to modify the contents of relations, but not of constant symbols. Similarly, in the linear algebra setting updates only affect matrix entry values, not matrix dimensions. An interesting line of future work is to consider dimension updates, which correspond to allow updates of constant symbols in CQs.

**Evaluation of q-hierarchical queries.** Similar than for free-connex queries, we can provide dynamic evaluation algorithms for qh-CQ and qh-MATLANG queries. However, for this dynamic setting, we require some additional algorithmic assumptions over the semiring. Let  $\mathcal{K} = (K, \oplus, \odot, \mathbf{0}, \mathbf{1})$  be a semiring and  $M$  be the set of all multisets of  $K$ . For any  $k \in K$  and  $m \in M$ , define  $\text{ins}(k, m)$  and  $\text{del}(k, m)$  to be the multisets resulting from inserting or deleting  $k$  from  $m$ , respectively. Then we say that  $\mathcal{K}$  is *sum-maintainable* if there exists a data structure  $\mathcal{D}$  to represent multisets of  $K$  such that the empty set  $\emptyset$  can be built in constant time, and if  $\mathcal{D}$  represents  $m \in M$  then: **(1)** the value  $\bigoplus_{k \in m} k$  can always be computed from  $\mathcal{D}$  in constant time; **(2)** a data structure that represents  $\text{ins}(k, m)$  can be obtained from  $\mathcal{D}$  in constant time; and **(3)** a data structure that represents  $\text{del}(k, m)$  can be obtained from  $\mathcal{D}$  in constant time. One can easily notice that if each element of  $\mathcal{K}$  has an additive inverse (i.e.,  $\mathcal{K}$  is a ring), then  $\mathcal{K}$  is sum-maintainable, like  $\mathbb{R}$ . Other examples of sum-maintainable semirings (without additive inverses) are  $\mathbb{B}$  and  $\mathbb{N}$ .

► **Theorem 15.** *Let  $\mathcal{K}$  be a sum-maintainable semi-integral domain. For every  $q$ -hierarchical CQ  $Q$ ,  $\text{DynEval}(Q, \sigma, \mathcal{K})$  can be evaluated dynamically with constant-time update and constant-delay. In particular,  $\text{DynEval}(\mathbf{Q}, \mathcal{S}, \mathcal{K})$  can also be evaluated dynamically with constant-time update and constant-delay for every qh-MATLANG  $\mathbf{Q}$  over  $\mathcal{S}$ .*

Similar than for free-connex CQ, we can extend the lower bound in [9] when the subsemiring generated by  $\mathbf{0}_{\mathcal{K}}$  and  $\mathbf{1}_{\mathcal{K}}$  is zero-sum free, by assuming the Online Boolean Matrix-Vector Multiplication (OMv) conjecture [29].

► **Theorem 16.** *Let  $Q$  be a CQ over  $\sigma$  without self-joins and constant-disjoint. Let  $\mathcal{K}$  be a semiring such that the subsemiring generated by  $\mathbf{0}_{\mathcal{K}}$  and  $\mathbf{1}_{\mathcal{K}}$  is zero-sum free. If  $\text{DynEval}(Q, \sigma, \mathcal{K})$  can be evaluated dynamically with constant-time update and constant-delay, then  $Q$  is  $q$ -hierarchical, unless the OMv conjecture is false.*

This transfers to conj-MATLANG, similarly to the lower bound in the free-connex case.

► **Corollary 17.** *Let  $\mathbf{Q}$  be a conj-MATLANG query over  $\mathcal{S}$  such that  $\mathbf{Q}$  does not repeat matrix symbols and  $\mathbf{Q}$  has guarded dimensions. Let  $\mathcal{K}$  be a semiring such that the subsemiring generated by  $\mathbf{0}_{\mathcal{K}}$  and  $\mathbf{1}_{\mathcal{K}}$  is zero-sum free. If  $\text{DynEval}(\mathbf{Q}, \mathcal{S}, \mathcal{K})$  can be evaluated dynamically with constant-time update and constant-delay, then  $\mathbf{Q}$  is equivalent to a qh-MATLANG query, unless the OMv conjecture is false.*

## 7 Conclusions and future work

In this work, we isolated the subfragments of conj-MATLANG that admit efficient evaluation in both static and dynamic scenarios. We found these algorithms by making the correspondence between CQ and MATLANG, extending the evaluation algorithms for free-connex and  $q$ -hierarchical CQ, and then translating these algorithms to the corresponding subfragments, namely, fc-MATLANG and qh-MATLANG. To the best of our knowledge, this is the first work that characterizes subfragments of linear algebra query languages that admit efficient evaluation. Moreover, this correspondence improves our understanding of its expressibility.

Regarding future work, a relevant direction is to extend fc-MATLANG and qh-MATLANG with disjunction, namely, matrix summation. This direction is still an open problem even for CQ with union [14]. Another natural extension is to add point-wise functions and understand how they affect expressibility and efficient evaluation. Finally, improving the lower bounds to queries without self-join would be interesting, which is also an open problem for CQ [8, 15].

---

### References

- 1 Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In Kimberly Keeton and Timothy Roscoe, editors, *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016*, pages 265–283. USENIX Association, 2016. URL: <https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi>.
- 2 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995. URL: <http://webdam.inria.fr/Alice/>.
- 3 Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.
- 4 Michael J. Anderson, Shaden Smith, Narayanan Sundaram, Mihai Capota, Zheguang Zhao, Subramanya Dullloor, Nadathur Satish, and Theodore L. Willke. Bridging the gap between HPC and big data frameworks. *Proc. VLDB Endow.*, 10(8):901–912, 2017. doi:10.14778/3090163.3090168.

- 5 Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. On acyclic conjunctive queries and constant delay enumeration. In Jacques Duparc and Thomas A. Henzinger, editors, *Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL, Lausanne, Switzerland, September 11-15, 2007, Proceedings*, volume 4646 of *Lecture Notes in Computer Science*, pages 208–222. Springer, 2007. doi:10.1007/978-3-540-74915-8\_18.
- 6 Pablo Barceló, Nelson Higuera, Jorge Pérez, and Bernardo Subercaseaux. On the expressiveness of LARA: A unified language for linear and relational algebra. In Carsten Lutz and Jean Christoph Jung, editors, *23rd International Conference on Database Theory, ICDT 2020, March 30-April 2, 2020, Copenhagen, Denmark*, volume 155 of *LIPICs*, pages 6:1–6:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.ICDT.2020.6.
- 7 Catriel Beeri, Ronald Fagin, David Maier, and Mihalis Yannakakis. On the desirability of acyclic database schemes. *J. ACM*, 30(3):479–513, 1983. doi:10.1145/2402.322389.
- 8 Christoph Berkholz, Fabian Gerhardt, and Nicole Schweikardt. Constant delay enumeration for conjunctive queries: A tutorial. *ACM SIGLOG News*, 7(1):4–33, 2020. doi:10.1145/3385634.3385636.
- 9 Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering conjunctive queries under updates. In Emanuel Sallinger, Jan Van den Bussche, and Floris Geerts, editors, *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, Chicago, IL, USA, May 14-19, 2017*, pages 303–318. ACM, 2017. doi:10.1145/3034786.3034789.
- 10 Philip A. Bernstein and Nathan Goodman. Power of natural semijoins. *SIAM J. Comput.*, 10(4):751–771, 1981. doi:10.1137/0210059.
- 11 Johann Brault-Baron. *De la pertinence de l'énumération: Complexité en logiques propositionnelle et du premier ordre. (The relevance of the list: propositional logic and complexity of the first order)*. PhD thesis, University of Caen Normandy, France, 2013. URL: <https://tel.archives-ouvertes.fr/tel-01081392>.
- 12 Robert Brijder, Floris Geerts, Jan Van den Bussche, and Timmy Weerwag. On the expressive power of query languages for matrices. *ACM Trans. Database Syst.*, 44(4):15:1–15:31, 2019. doi:10.1145/3331445.
- 13 Robert Brijder, Marc Gyssens, and Jan Van den Bussche. On matrices and k-relations. In Andreas Herzig and Juha Kontinen, editors, *Foundations of Information and Knowledge Systems - 11th International Symposium, FoIKS 2020, Dortmund, Germany, February 17-21, 2020, Proceedings*, volume 12012 of *Lecture Notes in Computer Science*, pages 42–57. Springer, 2020. doi:10.1007/978-3-030-39951-1\_3.
- 14 Nofar Carmeli and Markus Kröll. On the enumeration complexity of unions of conjunctive queries. *ACM Trans. Database Syst.*, 46(2):5:1–5:41, 2021. doi:10.1145/3450263.
- 15 Nofar Carmeli and Luc Segoufin. Conjunctive queries with self-joins, towards a fine-grained enumeration complexity analysis. In Floris Geerts, Hung Q. Ngo, and Stavros Sintos, editors, *Proceedings of the 42nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2023, Seattle, WA, USA, June 18-23, 2023*, pages 277–289. ACM, 2023. doi:10.1145/3584372.3588667.
- 16 Arnaud Durand and Etienne Grandjean. First-order queries on structures of bounded degree are computable with constant delay. *ACM Trans. Comput. Log.*, 8(4):21, 2007. doi:10.1145/1276920.1276923.
- 17 Arnaud Durand, Nicole Schweikardt, and Luc Segoufin. Enumerating answers to first-order queries over databases of low degree. In Richard Hull and Martin Grohe, editors, *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 121–131. ACM, 2014. doi:10.1145/2594538.2594539.
- 18 Idan Eldar, Nofar Carmeli, and Benny Kimelfeld. Direct access for answers to conjunctive queries with aggregation. *CoRR*, abs/2303.05327, 2023. doi:10.48550/arXiv.2303.05327.

- 19 Tarek Elgamal, Shangyu Luo, Matthias Boehm, Alexandre V. Evfimievski, Shirish Tatikonda, Berthold Reinwald, and Prithviraj Sen. SPOOF: sum-product optimization and operator fusion for large-scale machine learning. In *8th Biennial Conference on Innovative Data Systems Research, CIDR 2017, Chaminade, CA, USA, January 8-11, 2017, Online Proceedings*. www.cidrdb.org, 2017. URL: <http://cidrdb.org/cidr2017/papers/p3-elgamal-cidr17.pdf>.
- 20 Ahmed Elgohary, Matthias Boehm, Peter J. Haas, Frederick R. Reiss, and Berthold Reinwald. Scaling machine learning via compressed linear algebra. *SIGMOD Rec.*, 46(1):42–49, 2017. doi:10.1145/3093754.3093765.
- 21 Ronald Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *J. ACM*, 30(3):514–550, 1983. doi:10.1145/2402.322390.
- 22 Floris Geerts. On the expressive power of linear algebra on graphs. In Pablo Barceló and Marco Calautti, editors, *22nd International Conference on Database Theory, ICDT 2019, March 26-28, 2019, Lisbon, Portugal*, volume 127 of *LIPICs*, pages 7:1–7:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.ICDT.2019.7.
- 23 Floris Geerts, Thomas Muñoz, Cristian Riveros, and Domagoj Vrgoc. Expressive power of linear algebra query languages. In Leonid Libkin, Reinhard Pichler, and Paolo Guagliardo, editors, *Proceedings of the 40th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2021, Virtual Event, China, June 20-25, 2021*, pages 342–354. ACM, 2021. doi:10.1145/3452021.3458314.
- 24 Floris Geerts and Juan L. Reutter. Expressiveness and approximation properties of graph neural networks. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL: <https://openreview.net/forum?id=wIzUeM3TAU>.
- 25 Jonathan S Golan. *Semirings and their Applications*. Springer Science & Business Media, 2013. doi:10.1007/978-94-015-9333-5.
- 26 Todd J. Green, Gregory Karvounarakis, and Val Tannen. Provenance semirings. In Leonid Libkin, editor, *Proceedings of the 26th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS 2007, Beijing, China, June 11-13, 2007*, pages 31–40. ACM, 2007. doi:10.1145/1265530.1265535.
- 27 Udo Hebisch and Hanns Joachim Weinert. *Semirings: Algebraic theory and applications in computer science*, volume 5. World Scientific, 1998.
- 28 Udo Hebisch and Hans Joachim Weinert. Semirings and semifields. *Handbook of Algebra*, 1:425–462, 1996.
- 29 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 21–30. ACM, 2015. doi:10.1145/2746539.2746609.
- 30 Botong Huang, Shivnath Babu, and Jun Yang. Cumulon: Optimizing statistical data analysis in the cloud. In Kenneth A. Ross, Divesh Srivastava, and Dimitris Papadias, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013*, pages 1–12. ACM, 2013. doi:10.1145/2463676.2465273.
- 31 Muhammad Idris, Martín Ugarte, and Stijn Vansummeren. The dynamic Yannakakis algorithm: Compact and efficient query processing under updates. In Semih Salihoglu, Wenchao Zhou, Rada Chirkova, Jun Yang, and Dan Suciu, editors, *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, pages 1259–1274. ACM, 2017. doi:10.1145/3035918.3064027.
- 32 Muhammad Idris, Martín Ugarte, Stijn Vansummeren, Hannes Voigt, and Wolfgang Lehner. General dynamic Yannakakis: Conjunctive queries with theta joins under updates. *VLDB J.*, 29(2-3):619–653, 2020. doi:10.1007/S00778-019-00590-9.

- 33 Dimitrije Jankov, Shangyu Luo, Binhang Yuan, Zhuhua Cai, Jia Zou, Chris Jermaine, and Zekai J. Gao. Declarative recursive computation on an RDBMS: or, why you should use a database for distributed machine learning. *SIGMOD Rec.*, 49(1):43–50, 2020. doi:10.1145/3422648.3422659.
- 34 Konstantinos Kanellopoulos, Nandita Vijaykumar, Christina Giannoula, Roknoddin Azizi, Skanda Koppula, Nika Mansouri-Ghiasi, Taha Shahroodi, Juan Gómez-Luna, and Onur Mutlu. SMASH: co-designing software compression and hardware-accelerated indexing for efficient sparse matrix operations. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2019, Columbus, OH, USA, October 12-16, 2019*, pages 600–614. ACM, 2019. doi:10.1145/3352460.3358286.
- 35 Ahmet Kara, Milos Nikolic, Dan Olteanu, and Haozhe Zhang. Trade-offs in static and dynamic evaluation of hierarchical queries. In Dan Suciu, Yufei Tao, and Zhewei Wei, editors, *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2020, Portland, OR, USA, June 14-19, 2020*, pages 375–392. ACM, 2020. doi:10.1145/3375395.3387646.
- 36 Wojciech Kazana and Luc Segoufin. Enumeration of first-order queries on classes of structures with bounded expansion. In Richard Hull and Wenfei Fan, editors, *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA, June 22 - 27, 2013*, pages 297–308. ACM, 2013. doi:10.1145/2463664.2463667.
- 37 Shangyu Luo, Zekai J. Gao, Michael N. Gubanov, Luis Leopoldo Perez, and Christopher M. Jermaine. Scalable linear algebra on a relational database system. *SIGMOD Rec.*, 47(1):24–31, 2018. doi:10.1145/3277006.3277013.
- 38 Thomas Muñoz, Cristian Riveros, and Stijn Vansummeren. Enumeration and updates for conjunctive linear algebra queries through expressibility. *CoRR*, abs/2310.04118, 2023. doi:10.48550/arXiv.2310.04118.
- 39 Nicole Schweikardt, Luc Segoufin, and Alexandre Vigny. Enumeration for FO queries over nowhere dense graphs. In Jan Van den Bussche and Marcelo Arenas, editors, *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2018, Houston, TX, USA, June 10-15, 2018*, pages 151–163. ACM, 2018. doi:10.1145/3196959.3196971.
- 40 Amir Shaikhha, Mohammed Elseidy, Stephan Mihaila, Daniel Espino, and Christoph Koch. Synthesis of incremental linear algebra programs. *ACM Trans. Database Syst.*, 45(3):12:1–12:44, 2020. doi:10.1145/3385398.
- 41 Yisu Remy Wang, Shana Hutchison, Dan Suciu, Bill Howe, and Jonathan Leang. SPORES: sum-product optimization via relational equality saturation for large scale linear algebra. *Proc. VLDB Endow.*, 13(11):1919–1932, 2020. URL: <http://www.vldb.org/pvldb/vol13/p1919-wang.pdf>.
- 42 Fan Yang, Yuzhen Huang, Yunjian Zhao, Jinfeng Li, Guanxian Jiang, and James Cheng. The best of both worlds: Big data programming with both productivity and performance. In Semih Salihoglu, Wenchao Zhou, Rada Chirkova, Jun Yang, and Dan Suciu, editors, *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, pages 1619–1622. ACM, 2017. doi:10.1145/3035918.3058735.



# Direct Access for Conjunctive Queries with Negations

Florent Capelli  

Univ. Artois, CNRS, UMR 8188, Centre de Recherche en Informatique de Lens (CRIL),  
F-62300 Lens, France

Oliver Irwin  

Université de Lille, CNRS, Inria, UMR 9189 - CRISTAL, F-59000 Lille, France

---

## Abstract

Given a conjunctive query  $Q$  and a database  $\mathbf{D}$ , a direct access to the answers of  $Q$  over  $\mathbf{D}$  is the operation of returning, given an index  $j$ , the  $j^{\text{th}}$  answer for some order on its answers. While this problem is  $\#P$ -hard in general with respect to combined complexity, many conjunctive queries have an underlying structure that allows for a direct access to their answers for some lexicographical ordering that takes polylogarithmic time in the size of the database after a polynomial time precomputation. Previous work has precisely characterised the tractable classes and given fine-grained lower bounds on the precomputation time needed depending on the structure of the query. In this paper, we generalise these tractability results to the case of signed conjunctive queries, that is, conjunctive queries that may contain negative atoms. Our technique is based on a class of circuits that can represent relational data. We first show that this class supports tractable direct access after a polynomial time preprocessing. We then give bounds on the size of the circuit needed to represent the answer set of signed conjunctive queries depending on their structure. Both results combined together allow us to prove the tractability of direct access for a large class of conjunctive queries. On the one hand, we recover the known tractable classes from the literature in the case of positive conjunctive queries. On the other hand, we generalise and unify known tractability results about negative conjunctive queries – that is, queries having only negated atoms. In particular, we show that the class of  $\beta$ -acyclic negative conjunctive queries and the class of bounded nest set width negative conjunctive queries admit tractable direct access.

**2012 ACM Subject Classification** Information systems  $\rightarrow$  Relational database model

**Keywords and phrases** Conjunctive queries, factorized databases, direct access, hypertree decomposition

**Digital Object Identifier** 10.4230/LIPIcs.ICDT.2024.13

**Related Version** *Full Version*: <https://arxiv.org/abs/2310.15800> [10]

**Funding** This work was supported by project ANR KCODA, ANR-20-CE48-0004.

**Acknowledgements** We are thankful to Stefan Mengel and Sylvain Salvati for helpful discussions while elaborating these results.

## 1 Introduction

The *direct access (DA for short) task* is the problem of outputting, given  $k$ , the  $k$ -th answer of a query  $Q$  over a database  $\mathbf{D}$ . An error is returned if  $k$  is greater than the number of answers of  $Q$ . An order on  $\llbracket Q \rrbracket^{\mathbf{D}}$ , the answers of  $Q$  over  $\mathbf{D}$ , is assumed. This task has been introduced by Bagan, Durand, Grandjean and Olive in [2] and is very natural in the context of databases. It can be used as a building block for many other interesting tasks such as counting, enumerating [2] or sampling without repetition [13, 22] the answers of  $Q$ . Of course, if one has access to an ordered array containing  $\llbracket Q \rrbracket^{\mathbf{D}}$ , answering DA tasks simply consists in reading the right entry of the array. However, building such an array is often expensive,



© Florent Capelli and Oliver Irwin;  
licensed under Creative Commons License CC-BY 4.0  
27th International Conference on Database Theory (ICDT 2024).

Editors: Graham Cormode and Michael Shekelyan; Article No. 13; pp. 13:1–13:20



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

especially when the number of answers of  $Q$  is large. Hence, a natural approach for solving this problem is to simulate this method by using a data structure to represent  $\llbracket Q \rrbracket^{\mathbf{D}}$  that still allows for efficient DA tasks to be solved but that is cheaper to compute than the complete answer set. Moreover, one is rarely interested in accessing exactly one tuple of the answer set but one is usually interested in having a data structure allowing to solve efficiently DA tasks for any input  $k$ . Hence the time needed to solve exactly one DA task is not the best measure of complexity. To compare algorithms for DA tasks, it is then relevant to allow a *preprocessing phase* to construct a data structure that can be used later during the *access phase* where one needs to efficiently answer any DA task. Hence, we say that a method solves the DA problem if it consists in two algorithms  $P$  and  $A$  such that: on input  $Q$  and  $\mathbf{D}$ ,  $P$  constructs a data structure  $S$  and  $A(S, k)$  returns the  $k$ -th answer of  $Q$  for any  $k$ . To measure the quality of such an algorithm, we hence separate the *preprocessing time* – that is the time needed for executing  $P(Q, \mathbf{D})$  – and the *access time*, that is, the time needed to execute  $A(S, k)$ . For example, the method consisting in building an indexed array for  $\llbracket Q \rrbracket^{\mathbf{D}}$  solves DA with preprocessing time at least equal to the size of  $\llbracket Q \rrbracket^{\mathbf{D}}$  (and much higher in practice) and constant access time. While the access time is optimal in this case, the cost of preprocessing is often too high to pay in practice.

Previous work has consequently focused on devising methods with better preprocessing time while offering reasonable access time. In their seminal work [2], Bagan, Durand, Grandjean and Olive give a method for solving the DA problem with linear precomputation time and constant access time on a that works on the class of first order logic formulas and bounded degree databases. Bagan [1] later studied the problem for monadic second order formulas over bounded treewidth databases. Another line of research has focused on classes of conjunctive queries that admit efficient method for solving the DA problem. In [13], Carmeli, Zeevi, Berkholz, Kimelfeld, and Schweikardt give a method solving the DA problem for acyclic conjunctive queries with linear preprocessing time and polylogarithmic access time for a well-chosen lexicographical order. The results also hold for bounded fractional hypertree width queries, a number measuring how far a conjunctive query is from being acyclic. It generalises many results from the seminal paper by Yannakakis establishing the tractability of testing non-emptiness of acyclic conjunctive queries [33] to the tractability of counting the number of answers of conjunctive queries [30] having bounded hypertree width. later improved this result by precisely characterising the lexicographical ordering allowing for this kind of complexity guarantees. Fine-grained characterisation of the complexity of solving the DA problem on conjunctive queries, whose answers are assumed to be ordered by some lexicographical order, has been given by Carmeli, Tziavelis, Gatterbauer, Kimelfeld and Riedewald in [12] for the case of acyclic queries and by Bringmann, Carmeli and Mengel in [7] for the general case. Recently, Eldar, Carmeli and Kimelfeld [15] studied the complexity of solving the DA problem for conjunctive queries with aggregation.

In this paper, we devise new methods for solving the DA problem for *signed conjunctive queries*, that is, conjunctive queries that may contain negated atoms. This is particularly challenging because only a few tractability results are known on signed conjunctive queries. The problem of testing non-emptiness of signed conjunctive queries being NP-hard on acyclic conjunctive queries with respect to combined complexity, it is not possible to directly build on the work cited in the last paragraph. Two classes of negative conjunctive queries (that is, conjunctive queries where every atom is negated) have been shown so far to support efficient non-emptiness testing: the class of  $\beta$ -acyclic queries [29, 4] and the class of bounded nested-set width queries [25]. The former has been shown to also support efficient (weighted) counting [6, 9]. Our main contribution is a generalisation of these results to DA. More



precisely, we give a method that efficiently solves the DA problem on a large class of signed conjunctive queries, which contains in particular  $\beta$ -acyclic negative conjunctive queries, bounded nest-width negative conjunctive queries and bounded fractional hypertree width positive conjunctive queries. For the latter case, the complexity we obtain is similar to the one presented in [7] and we also get complexity guarantees depending on a lexicographical ordering that can be specified by the user. Hence our result both improves the understanding of the tractability of signed conjunctive queries and unifies the existing results with the positive case. In a nutshell, we prove that the complexity of solving the DA problem for a lexicographical order of a signed conjunctive query  $Q$  roughly matches the complexity proven in [7] for the worst positive query we could construct by removing some negative atoms of  $Q$  and turning the remaining ones to positive atoms. It is not surprising that the complexity of solving the DA problem on signed conjunctive queries depends on the complexity of solving the DA problem for subqueries since one could simulate direct access to such a subquery by choosing a database where the removed negated atoms are associated with empty relations, hence, making them virtually useless in the query. However, proving the actual combined complexity upper bound is not trivial to obtain and necessitates introducing new tools to handle negated atoms.

As a byproduct, we introduce a new notion of hypergraph width based on elimination order, the  $\beta$ -hyperorder width, that is hereditary – in the sense that the width of every subhypergraph does not exceed the width of the original hypergraph – which makes it particularly well tailored for studying the tractability of negative conjunctive queries. We show that this notion sits between nest-set width and  $\beta$ -hypertree width [18], but do not suffer from one important drawback of working with  $\beta$ -hypertree width: our width notion is based on a decomposition that works for every subhypergraph.

Our method is based on a two-step preprocessing. Given a signed conjunctive query  $Q$ , a database  $\mathbf{D}$  and an order  $\prec$  on its variables, we start by constructing a circuit which computes  $\llbracket Q \rrbracket^{\mathbf{D}}$  in a factorised way enjoying interesting syntactical properties. The size of this circuit depends on the complexity of the order  $\prec$  chosen on the variables of  $Q$ . We then show that, with a second light preprocessing on the circuit itself, we can answer DA tasks for the lexicographical order induced by  $\prec$  on the circuit in time  $\text{poly}(n)\text{polylog}(D)$  where  $n$  is the number of variables of  $Q$  and  $D$  is the domain of  $\mathbf{D}$ . This approach is akin to the approach used in *factorised databases* introduced by Olteanu and Závodný [27], a fruitful approach allowing efficient management of the answer set of a query by working directly on a factorised representation of the answer set instead of working on the query itself [26, 32, 3, 28]. However, the restrictions that we are considering in this paper are different from the one used in previous work since we need somehow to account for the variable ordering in the circuit itself. The syntactic restrictions we use have already been considered in [9] where they are useful to deal with  $\beta$ -acyclic CNF formulas.

The paper is organised as follows: Section 2 introduces the notations and concepts necessary to understand the paper. We then present the family of circuits we use to represent database relations and the DA method for circuits in Section 3. Section 4 presents the algorithm used to construct a circuit representing  $\llbracket Q \rrbracket^{\mathbf{D}}$  from a join query  $Q$  (that is a conjunctive query without existential quantifiers) and a database  $\mathbf{D}$ . Upper bounds on the size of the circuits are given in Section 4.3 using hypergraph decompositions defined in Section 4.2. Finally Section 5 explicitly states the results we obtain by combining both techniques together, explains how one can go from join query to conjunctive query by existentially projecting variables directly in the circuit and makes connections with the existing literature. We conclude with interesting research directions in Section 6.

Due to space restriction, full proofs have been omitted from the paper. Proofs of theorems and lemmas tagged with a star symbol  $\star$  can be found in the full version of this paper [10].

## 2 Preliminaries

Given  $n \in \mathbb{N}$ , we denote by  $[n]$  the set  $\{0, \dots, n\}$ . When writing down complexity, we use the notation  $\text{poly}(n)$  to denote that the complexity is polynomial in  $n$ ,  $\text{poly}_k(n)$  to denote that the complexity is polynomial in  $n$  when  $k$  is considered a constant (in other words, the coefficients and the degree of the polynomial may depend on  $k$ ) and  $\text{polylog}(n)$  to denote that the complexity is polynomial in  $\log(n)$ .

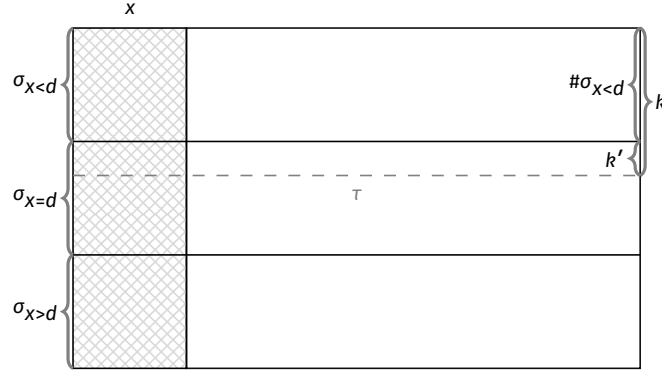
**Tuples and relations.** Let  $D$  and  $X$  be finite sets. A (named) *tuple* on domain  $D$  and variables  $X$  is a mapping from  $X$  to  $D$ . We denote by  $D^X$  the set of all tuples on domain  $D$  and variables  $X$ . A *relation*  $R$  on domain  $D$  and variables  $X$  is a subset of tuples, that is,  $R \subseteq D^X$ . Given a tuple  $\tau \in D^X$  and  $Y \subseteq X$ , we denote by  $\tau|_Y$  the tuple on domain  $D$  and variable  $Y$  such that  $\tau|_Y(y) = \tau(y)$  for every  $y \in Y$ . Given a variable  $x \in X$  and  $d \in D$ , we denote by  $[x \leftarrow d]$  the tuple on variables  $\{x\}$  that assigns the value  $d \in D$  to  $x$ . We denote by  $\langle \rangle$  the empty tuple, that is, the only element of  $D^\emptyset$ . Given two tuples  $\tau_1 \in D^{X_1}$  and  $\tau_2 \in D^{X_2}$ , we say that  $\tau_1$  and  $\tau_2$  are *compatible*, denoted by  $\tau_1 \simeq \tau_2$ , if  $\tau_1|_{X_1 \cap X_2} = \tau_2|_{X_1 \cap X_2}$ . In this case, we write  $\tau_1 \bowtie \tau_2$  the tuple on domain  $D$  and variables  $X_1 \cup X_2$  defined as  $(\tau_1 \bowtie \tau_2)(x)$  to be  $\tau_1(x)$  if  $x \in X_1$  and  $\tau_2(x)$  otherwise. If  $X_1 \cap X_2 = \emptyset$ , we write  $\tau_1 \times \tau_2$ . The *join*  $R_1 \bowtie R_2$  of  $R_1$  and  $R_2$ , for two relations  $R_1, R_2$  on domain  $D$  and variables  $X_1, X_2$  respectively, is defined as  $\{\tau_1 \bowtie \tau_2 \mid \tau_1 \in R_1, \tau_2 \in R_2, \tau_1 \simeq \tau_2\}$ . Observe that if  $X_1 \cap X_2 = \emptyset$ ,  $R_1 \bowtie R_2$  is simply the *Cartesian product* of  $R_1$  and  $R_2$ . Then, we denote it by  $R_1 \times R_2$ .

Let  $R \subseteq D^X$  be a relation from a set of variables  $X$  to a domain  $D$ . We denote  $\sigma_F(R)$  as the subset of  $R$  where the formula  $F$  is true. Throughout the paper, we will assume that both the domain  $D$  and the variable set  $X$  are ordered. The order on  $D$  will be denoted as  $<$  and the order on  $X$  as  $\prec$  and we will often write  $D = \{d_1, \dots, d_p\}$  with  $d_1 < \dots < d_p$  and  $X = \{x_1, \dots, x_n\}$  with  $x_1 \prec \dots \prec x_n$ . Given  $d \in D$ , we denote by  $\text{rank}(d)$  the number of elements of  $D$  that are smaller or equal to  $d$ . Both  $<$  and  $\prec$  induce a lexicographical order  $\prec_{\text{lex}}$  on  $D^X$  defined as  $\tau \prec_{\text{lex}} \sigma$  if there exists  $x \in X$  such that for every  $y \prec x$ ,  $\tau(y) = \sigma(y)$  and  $\tau(x) < \sigma(x)$ . Given an integer  $k \leq \#R$ , we denote by  $R[k]$  the  $k^{\text{th}}$  tuple in  $R$  for the  $\prec_{\text{lex}}$ -order. The following observation will prove useful to design a DA algorithm:

► **Lemma 1** ( $\star$ ). *Let  $\tau = R[k]$  and  $x = \min(\text{var}(R))$ . Then  $\tau(x) = \min\{d \mid \#\sigma_{x \leq d}(R) \geq k\}$ . Moreover,  $\tau = R'[k']$ , where  $R' = \sigma_{x=d}(R)$  and  $k' = k - \#\sigma_{x < d}(R)$ .*

Figure 1 gives an intuition of the result presented in Lemma 1 as a visual representation of the index transformation.

**Queries.** A *positive atom* (resp. *negative atom*) is an expression of  $R(\mathbf{x})$  (resp.  $\neg R(\mathbf{x})$ ) where  $R$  is a relation symbol and  $\mathbf{x}$  a tuple of variables in  $X$ . A *signed join query*  $Q$  is a set of (negative or positive) atoms  $Q = \{R_1(\mathbf{x}_1), \dots, R_p(\mathbf{x}_p), \neg S_{p+1}(\mathbf{x}_{p+1}), \dots, \neg S_m(\mathbf{x}_m)\}$ . In this paper, we consider *self-join free* queries, that is, we assume that two distinct atoms of a join query have distinct relation symbols. The set of variables of  $Q$  is denoted by  $\text{var}(Q)$ , the set of positive (resp. negative) atoms of  $Q$  is denoted by  $\text{atoms}^+(Q)$  (resp.  $\text{atoms}^-(Q)$ ). A *positive* (resp. *negative*) *join query* is a signed join query without negative (resp. positive) atoms. The size  $|Q|$  of  $Q$  is defined as  $\sum_{i=1}^m |\mathbf{x}_i|$ , where  $|\mathbf{x}|$  denotes the number of variables in  $\mathbf{x}$ . A *database*  $\mathbf{D}$  for  $Q$  is an ordered finite set  $D$  called the *domain* together with a set of



■ **Figure 1** Representation of the link between  $k$  and  $k'$ .

relations  $R_i^{\mathbf{D}} \subseteq D^{a_i}$ ,  $S_j^{\mathbf{D}} \subseteq D^{a_j}$  such that  $a_i = |\mathbf{x}_i|$ . The *answers of  $Q$  over  $\mathbf{D}$*  is the relation  $\llbracket Q \rrbracket^{\mathbf{D}} \subseteq D^{\text{var}(Q)}$  defined as the set of  $\sigma \in D^{\text{var}(Q)}$  such that for every  $i \leq p$ ,  $\sigma(\mathbf{x}_i) \in R_i^{\mathbf{D}}$  and for every  $p < i \leq m$ ,  $\sigma(\mathbf{x}_i) \notin S_i^{\mathbf{D}}$ . The size  $|\mathbf{D}|$  of the database  $\mathbf{D}$  is defined to be the total number of tuples in it plus the size of its domain, that is,  $|\mathbf{D}| + \sum_{i=1}^{\ell} |R_i| + \sum_{i=\ell+1}^m |S_i|$ . We follow the definition of [25] about the size of the database. Adding the size of the domain here is essential since we are dealing with negative atoms. Hence a query may have answers even when the database is empty, for example,  $Q = \neg R(x)$  with  $R^{\mathbf{D}} = \emptyset$  has  $|\mathbf{D}|$  answers.

A *signed conjunctive query*  $Q(Y)$  is a join query  $Q$  together with  $Y \subseteq \text{var}(Q)$ , called the *free variables of  $Q$*  and denoted by  $\text{free}(Q)$ . The answers  $\llbracket Q(Y) \rrbracket^{\mathbf{D}}$  of a conjunctive query  $Q$  over a database  $\mathbf{D}$  are defined as  $\llbracket Q \rrbracket^{\mathbf{D}}|_Y$ , that is, the projection over  $Y$  of answers of  $Q$ .

**Direct Access tasks.** Given a query  $Q$ , a database instance  $\mathbf{D}$  on an ordered domain  $D$  and a total order  $\prec$  on the variables of  $Q$ , a *Direct Access (DA for short) task* [12] is the problem of returning, on input  $k$ , the  $k$ -th tuple  $\llbracket Q \rrbracket^{\mathbf{D}}[k]$  for the order  $\prec|_{\text{lex}}$  if  $k < \#\llbracket Q \rrbracket^{\mathbf{D}}$  and fail otherwise. We are interested in answering DA tasks using the same setting as [12]: we allow a *precomputation* phase during which a data structure is constructed, followed by an *access* phase. Our goal is to obtain an algorithm for DA tasks with polynomial precomputation time and access time that is polylogarithmic time in the size of  $\mathbf{D}$ .

**Hypergraphs and Signed Hypergraphs.** A *hypergraph*  $H = (V, E)$  is defined as a set of *vertices*  $V$  and *hyperedges*  $E \subseteq 2^V$ , that is, a hyperedge  $e \in E$  is a subset of  $V$ . A *signed hypergraph*  $H = (V, E_+, E_-)$  is defined as a set of *vertices*  $V$ , *positive edges*  $E_+ \subseteq 2^V$  and *negative edges*  $E_- \subseteq 2^V$ . The *signed hypergraph*  $H(Q) = (\text{var}(Q), E_+, E_-)$  of a *signed conjunctive query*  $Q(Y)$  is defined as the signed hypergraph whose vertex set is the variables of  $Q$  and such that  $E_+ = \{\text{var}(a) \mid a \text{ is a positive atom of } Q\}$  and  $E_- = \{\text{var}(a) \mid a \text{ is a negative atom of } Q\}$ .

A *subhypergraph*  $H'$  of  $H$ , denoted by  $H' \subseteq H$  is a hypergraph of the form  $(V, E')$  with  $E' \subseteq E$ . For  $S \subseteq V$ , we denote by  $H \setminus S$  the hypergraph  $(V \setminus S, E')$  where  $E' = \{e \setminus S \mid e \in E\}$ . Given  $v \in V$ , we denote by  $E(v) = \{e \in E \mid v \in e\}$  the set of edges containing  $v$ , by  $N_H(v) = \bigcup_{e \in E(v)} e$  the *neighbourhood of  $v$  in  $H$*  and by  $N_H^*(v) = N_H(v) \setminus \{v\}$  the *open neighbourhood of  $v$* . We will be interested in the following vertex removal operation on  $H$ : given a vertex  $v$  of  $H$ , we denote by  $H/v = (V \setminus \{v\}, E/v)$  where  $E/v$  is defined as  $\{e \setminus \{v\} \mid e \in E\} \setminus \{\emptyset\} \cup N_H^*(v)$ , that is,  $H/v$  is obtained from  $H$  by removing  $v$  from every edge of  $H$  and by adding a new edge that contains the open neighbourhood of  $v$ .

## 13:6 Direct Access for Conjunctive Queries with Negations

Given  $S \subseteq V$  and  $E \subseteq 2^V$ , a *covering of  $S$  with  $E$*  is a subset  $F \subseteq E$  such that  $S \subseteq \bigcup_{e \in F} e$ . The *cover number  $cn(S, E)$  of  $S$  wrt  $E$*  is defined as the minimal size of a covering of  $S$  with  $E$ , that is,  $cn(S, E) = \min\{|F| \mid F \text{ is a covering of } S \text{ with } E\}$ . A *fractional covering of  $S$  with  $E$*  is a function  $c : E \rightarrow \mathbb{R}_+$  such that for every  $s \in S$ ,  $\sum_{e \in E(s)} c(e) \geq 1$ . Observe that a covering is a fractional covering where  $c$  has values in  $\{0, 1\}$ . The *fractional cover number  $fcn(S, E)$  of  $S$  wrt  $E$*  is defined as the minimal size of a fractional covering of  $S$  with  $E$ , that is,  $fcn(S, E) = \min\{\sum_{e \in E} c(e) \mid c \text{ is a fractional covering of } S \text{ with } E\}$ . Fractional covers are particularly interesting because of the following theorem by Grohe and Marx:

► **Theorem 2** ([19]). *Let  $Q$  be a join query and  $\lambda$  be the fractional cover number of  $\text{var}(Q)$ . Then for every database  $\mathbf{D}$ ,  $\llbracket Q \rrbracket^{\mathbf{D}}$  has size at most  $|\mathbf{D}|^\lambda$ .*

### 3 Ordered relational circuits

In this section, we introduce a data structure that can be used to succinctly represent relations. This data structure is an example of a factorised representation, such as d-representations [28], but does not need to be structured along a tree, which will allow us to handle more queries, and especially queries with negative atoms – for example  $\beta$ -acyclic signed conjunctive queries, a class of queries that do not have polynomial size d-representations [9, Theorem 9].

#### 3.1 Definitions

**Relational circuits.** A  $\{\bowtie, \text{dec}\}$ -circuit  $C$  on variables  $X = \{x_1, \dots, x_n\}$  and domain  $D$  is a multi-directed acyclic graph, that is, there may be more than one edge between two nodes  $u$  and  $v$ , with one special gate  $\text{out}(C)$  called the *output* of  $C$ . The circuit is labelled as follows:

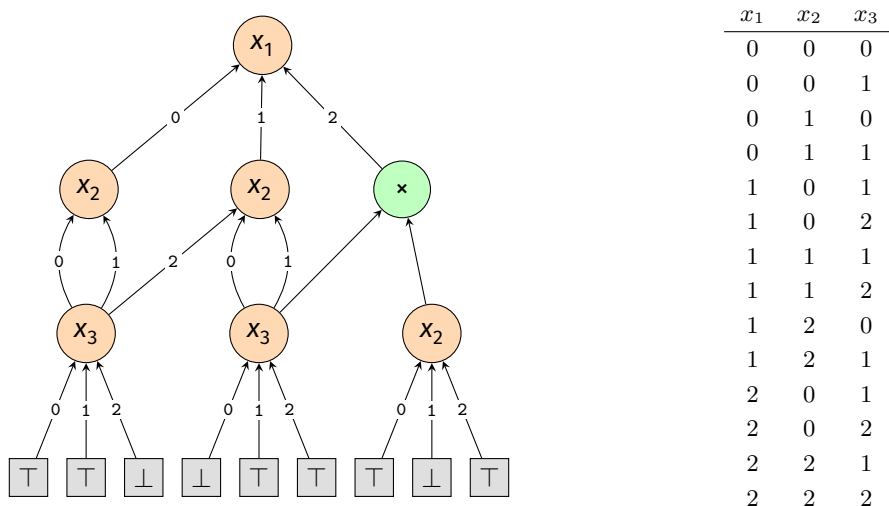
- every gate of  $C$  with no ingoing edge, called an *input* of  $C$ , is labelled by either  $\perp$  or  $\top$ ;
- a gate  $v$  labelled by a variable  $x \in X$  is called a *decision gate*. We denote  $x$  by  $\text{decvar}(v)$ . Each ingoing edge  $e$  of  $v$  is labelled by a value  $d \in D$  and for each  $d \in D$ , there is at most one ingoing edge of  $v$  labelled by  $d$ . This implies that a decision gate has at most  $|D|$  ingoing edges; and
- every other gate is labelled by  $\bowtie$ .

The set of all the decision gates in a circuit  $C$  is denoted by  $\text{decision}(C)$ . Given a gate  $v$  of  $C$ , we denote by  $C_v$  the *subcircuit of  $C$  rooted in  $v$*  to be the circuit whose gates are the gates reachable from  $v$  by following a directed path in  $C$ . We define the *variable set of  $v$* , denoted by  $\text{var}(v) \subseteq X$ , to be the set of variables  $x$  labelling a decision gate in  $C_v$ . The *size*  $|C|$  of a  $\{\bowtie, \text{dec}\}$ -circuit is defined to be the number of edges of its underlying DAG. In the example circuit of Figure 2, the decision gates are represented as containing the variable that labels them. Considering  $v$  to be the leftmost  $x_2$  decision gate, we have  $\text{var}(v) = \{x_2, x_3\}$ .

We define the *relation  $\text{rel}(v) \subseteq D^{\text{var}(v)}$  computed at gate  $v$*  inductively as follows: if  $v$  is an input labelled by  $\perp$ , then  $\text{rel}(v) = \emptyset$ . If  $v$  is an input labelled by  $\top$ , then  $\text{rel}(v) = D^\emptyset$ , that is,  $\text{rel}(v)$  is the relation containing only the empty tuple. Otherwise, let  $v_1, \dots, v_k$  be the inputs of  $v$ . If  $v$  is a  $\bowtie$ -gate, then  $\text{rel}(v)$  is defined to be  $\text{rel}(v_1) \bowtie \dots \bowtie \text{rel}(v_k)$ . If  $v$  is a decision gate labelled by a variable  $x$ ,  $\text{rel}(v) = ([x \leftarrow d_1] \bowtie \text{rel}(v_1) \times D^{\Delta(v, v_1)}) \cup \dots \cup ([x \leftarrow d_k] \bowtie \text{rel}(v_k) \times D^{\Delta(v, v_k)})$  where  $e_i$  is the incoming edge  $(v_i, v)$  labelled by  $d_i$  and  $\Delta(v, v_i) = \text{var}(v) \setminus (\{x\} \cup \text{var}(v_i))$ . It is readily verified that  $\text{rel}(v)$  is a relation on domain  $D$  and variables  $\text{var}(v)$ . The *relation computed by  $C$  over a set of variables  $X$*  (assuming  $\text{var}(C) \subseteq X$ ), denoted by  $\text{rel}_X(C)$ , is defined to be  $\text{rel}(\text{out}(C)) \times D^{X \setminus \text{var}(\text{out}(C))}$ . In Figure 2, if  $v$  is the leftmost  $x_3$  decision gate, then  $\text{rel}(v)$  is the relation where  $x_3$  is set to 0 or 1.

Deciding whether the relation computed by a  $\{\bowtie, \text{dec}\}$ -circuit is non-empty is NP-complete by a straightforward reduction to testing non-emptiness of conjunctive queries [14]. Such circuits are hence of little use to get tractability results. We are therefore more interested in the following restriction of  $\{\bowtie, \text{dec}\}$ -circuits where testing non-emptiness and other related tasks are tractable: a  $\{\times, \text{dec}\}$ -circuit  $C$  is a  $\{\bowtie, \text{dec}\}$ -circuit such that: (i) for every  $\bowtie$ -gate  $v$  of  $C$  with inputs  $v_1, \dots, v_k$  and  $i < j \leq k$ , it holds that  $\text{var}(v_i) \cap \text{var}(v_j) = \emptyset$ , (ii) for every decision gate  $v$  of  $C$  labelled by  $x$  with inputs  $v_1, \dots, v_k$  and  $i \leq k$ , it holds that  $x \notin \text{var}(v_i)$ . An example of such a circuit is presented in Figure 2.

Let  $X$  be a set of variables ordered by  $\prec$ . A  $\{\times, \text{dec}\}$ -circuit  $C$  on domain  $D$  and variables  $X$  is a  $\prec$ -ordered  $\{\times, \text{dec}\}$ -circuit if for every decision gate  $v$  of  $C$  labelled with  $x \in X$ , it holds that for every  $y \in \text{var}(v) \setminus \{x\}$ ,  $x \prec y$ . We simply say that a circuit  $C$  is an ordered  $\{\times, \text{dec}\}$ -circuit if there exists some order  $\prec$  on  $X$  such that  $C$  is a  $\prec$ -ordered  $\{\times, \text{dec}\}$ -circuit. Observe that if  $v$  is a decision-gate in an ordered  $\{\times, \text{dec}\}$ -circuit, then  $\text{rel}(v) = ([x \leftarrow d_1] \times \text{rel}(v_1) \times D^{\Delta(v, v_1)}) \uplus \dots \uplus ([x \leftarrow d_k] \times \text{rel}(v_k) \times D^{\Delta(v, v_k)})$  since  $x \notin \text{var}(v_i)$  by definition and that these unions are disjoint because of the value assigned to  $x$ . The circuit presented in Figure 2 is an ordered  $\{\times, \text{dec}\}$ -circuit for the order induced by the variables  $x_1, x_2, x_3$ .

(a) ordered  $\{\times, \text{dec}\}$ -circuit.

(b) relation computed by the circuit.

■ **Figure 2** Example of a small ordered  $\{\times, \text{dec}\}$ -circuit computing a simple relation.

**Frontiers.** Let  $X = \{x_1, \dots, x_n\}$  with  $x_1 \prec \dots \prec x_n$ . A *prefix assignment* of size  $p$  is an assignment of variables  $\tau \in D^{\{x_1, \dots, x_p\}}$  with  $p \leq n$ . When answering DA tasks, we will need to be able to find a representation of the tuples in  $\text{rel}(C)$  that agree with  $\tau$ . To do that, we introduce the notion of frontier. Given  $\tau$  a prefix assignment, the *frontier*  $f_\tau$  of  $\tau$  in  $C$  is defined algorithmically as follows:

1. Instantiate a set  $F$  with  $\text{out}(C)$ , the root of the circuit.
2. As long as  $F$  is not stable, do:
  - if  $v \in F$  is a  $\times$ -gate,  $F := (F \setminus \{v\}) \cup \bigcup_{w \in \text{input}(v)} w$ ,
  - if  $v \in F$  is a decision gate and the variable  $x$  labelling  $v$  is assigned in the prefix ( $x \in \{x_1, \dots, x_p\}$ ),  $F := (F \setminus \{v\}) \cup \{v'\}$  where  $v'$  is the node connected to  $v$  by the edge  $(v', v)$  labelled by  $\tau(x)$ .
3. If  $F$  contains a  $\perp$ -gate, then  $f_\tau = \{\perp\}$ , otherwise  $f_\tau = F$ .

## 13:8 Direct Access for Conjunctive Queries with Negations

Frontiers are particularly useful because they can be efficiently computed and the relation they represent is essentially the tuples of the relation represented by  $C$  that agree with  $\tau$ . We denote by  $\text{var}(f_\tau) = \bigcup_{v \in f_\tau} \text{var}(v)$  the set of variables appearing below a gate in the frontier and by  $\text{rel}(f_\tau) = \bigtimes_{v \in f_\tau} \text{rel}(v)$ . Given a prefix assignment  $\tau$  of variables  $\{x_1, \dots, x_p\}$  and a relation  $R$ , we slightly abuse notation by denoting  $\sigma_\tau(R)$  the relation  $\sigma_{x_1=\tau(x_1) \wedge \dots \wedge x_p=\tau(x_p)}(R)$ . We can prove by induction on  $p$  that:

► **Lemma 3** ( $\star$ ). *Let  $\tau$  be a prefix assignment on variables  $x_1, \dots, x_p$ . Then  $f_\tau$  can be computed in  $\mathcal{O}(|X|)$  and  $\sigma_\tau(\text{rel}_X(C)) = \{\tau\} \times \text{rel}(f_\tau) \times D^{\{x_{p+1}, \dots, x_n\} \setminus \text{var}(f_\tau)}$ .*

### 3.2 Direct Access for ordered $\{\times, \text{dec}\}$ -circuits

The main result of this section is an algorithm that allows for efficient solving of DA tasks for an ordered  $\{\times, \text{dec}\}$ -circuit on domain  $D$  and variables  $X$ . More precisely, we prove:

► **Theorem 4**. *Let  $(X, \prec)$  be a finite ordered set and  $C$  a  $\prec$ -ordered  $\{\times, \text{dec}\}$ -circuit on variables  $X$  and domain  $D$ . We can answer direct access tasks on  $\text{rel}_X(C)$  ordered by  $\prec_{\text{lex}}$  with precomputation time  $\mathcal{O}(|C| \cdot \text{poly}(|X|) \cdot \text{polylog}|D|)$  and access time  $\mathcal{O}(\text{poly}(|X|) \cdot \text{polylog}|D|)$ .*

**Precomputation.** In this section, we assume that  $C$  is a  $\prec$ -ordered  $\{\times, \text{dec}\}$ -circuit on variables  $X$ . The *count label* of  $C$ , denoted by  $\text{nrel}_C$ , is the mapping from  $\text{decision}(C) \times D$  to  $\mathbb{N}$  such that  $\text{nrel}_C(v, d) = \#\sigma_{x \leq d}(\text{rel}(v))$ , that is,  $\text{nrel}_C(v, d)$  is the number of tuples from  $\text{rel}(v)$  that assign a value on  $x$  smaller or equal than  $d$ . The precomputation step aims to compute  $\text{nrel}_C$  so that we can access  $\text{nrel}_C(v, d)$  quickly.

Our algorithm performs a bottom-up computation of the number of satisfying tuples in  $\text{rel}(v)$  for every gate  $v$  of  $C$ . If  $\text{rel}(v)$  is a decision-gate on variable  $x$ , we have by definition:

$$|\text{rel}(v)| = \sum_{w \in \text{input}(v)} |\text{rel}(w)| \times |D|^{|\Delta(v,w)|} \text{ where } \Delta(v,w) = \text{var}(v) \setminus (\{x\} \cup \text{var}(w)). \quad (1)$$

Similarly,  $\text{nrel}_C(v, d)$  can be computed by restricting the previous relation on the inputs of  $v$  that sets  $x$  to a value  $d' \leq d$ , that is:

$$\text{nrel}_C(v, d) = \sum_{w \in \text{input}(v), d_w \leq d} |\text{rel}(w)| \times |D|^{|\Delta(v,w)|} \text{ where } d_w \text{ is the label of edge } (v, w). \quad (2)$$

Finally observe that if  $v$  is a  $\times$ -gate, we clearly have  $|\text{rel}(v)| = \prod_{w \in \text{input}(v)} |\text{rel}(w)|$  and for every gate  $v$ ,  $\text{var}(v) = \bigcup_{w \in \text{input}(v)} \text{var}(w)$ .

Hence, using a dynamic programming algorithm that follows the structure of the circuit in a bottom-up way, we can compute tables  $T_{\text{rel}}$ ,  $T_{\text{var}}$  and  $T_{\text{nrel}_C}$  such that  $T_{\text{rel}}[v] = |\text{rel}(v)|$ ,  $T_{\text{var}}[v] = \text{var}(v)$  and  $T_{\text{nrel}_C}[v, d] = \text{nrel}_C(v, d)$  for every gate  $v$  and value  $d \in D$ , allowing us to prove the following (a full description of the dynamic programming algorithm can be found in the full version):

► **Lemma 5** ( $\star$ ). *Given a  $\prec$ -ordered  $\{\times, \text{dec}\}$ -circuit  $C$ , we can compute a data structure in time  $\mathcal{O}(|C| \cdot \text{poly}|X| \text{polylog}|D|)$  that allows us to access  $\text{var}(v)$ ,  $|\text{rel}(v)|$  for every gate  $v$  of  $C$  in time  $\mathcal{O}(1)$  and  $\text{nrel}_C(v, d)$  for every decision gate  $v$  and  $d \in D$  in time  $\mathcal{O}(\text{polylog}(|D|))$ .*

An example of what the  $\text{nrel}_C$  values might look like in practice can be seen at the left of the decision gates in Figure 3.



**Direct access.** We now show how the precomputation from Lemma 5 allows us to get direct access for ordered  $\{\times, \text{dec}\}$ -circuits. We first show how one can solve a DA task for any relation as long as we have access to very simple counting oracles. We then show that one can quickly simulate these oracle calls in ordered  $\{\times, \text{dec}\}$ -circuits using precomputed values.

We start by illustrating our algorithm on an example.

► **Example 6.** Consider Figure 3, which represents two different direct access tasks on a circuit where  $\text{nrel}_C$  has been precomputed and which is represented as lists beside each node.

We start by explaining how we solve direct access for  $k = 7$ , depicted in Figure 3a. In this paragraph, let  $\tau_7$  be the 7-th solution of the circuit, that is,  $\{x_1 \mapsto 1, x_2 \mapsto 1, x_3 \mapsto 1\}$  (see Figure 2). Our algorithm iteratively finds the values of  $\tau_7$  on  $x_1, x_2$  and  $x_3$ . It starts by finding the value  $\tau_7(x_1)$  from the decision node at the root of the circuit. Using  $\text{nrel}_C$ , one can see that the smallest value one can assign to  $x_1$  to have at least 7 solutions is 1. By Lemma 1, we know that  $\tau_7(x_1) = 1$ . The algorithm then follows the edge labelled by 1 to find a subcircuit computing every solution of the circuit when  $x_1$  is fixed to 1. We can now repeat the method to find  $\tau_7(x_2)$ . However, one has to be careful about the index. By setting  $x_1$  to 1, we have discarded every solution of the circuit where  $x_1 < 1$ , that is, every solution where  $x_1 \leq 0$ . From  $\text{nrel}_C$ , we know that there are 4 such solutions. Hence, when repeating the method, we now look for the third ( $7 - 4 = 3$ ) solution of the next gate. The same reasoning yields that  $\tau_7(x_2) = 1$  and that we discard 2 solutions. Applying this method once more yields the desired value for  $x_3$ .

The algorithm is a bit more complex when encountering  $\times$ -gates. We illustrate it in Figure 3b, where we solve direct access for  $k = 13$ . Again, we denote by  $\tau_{13}$  the 13-th solution of the circuit. We find that  $\tau_{13}(x_1) = 2$  exactly as before. We know that setting  $x_1$  to 2 discards 10 solutions. Hence, we need to find the 3-rd solution of the circuit rooted in the  $\times$ -gate. To do that, we first follow every edge going in the gate to find only decision-gates. By 3, we know that the solution of the  $\times$ -gate are the Cartesian product of the solutions of both decision-gates. Now to find the value  $\tau_{13}(x_2)$ , we do the same reasoning as before but one has to be careful:  $\text{nrel}_C$  only contains the number of solutions of the subcircuit but each one of these solutions can be extended to a full solution of the circuit by completing it with the value of  $x_3$ . We now from  $\text{nrel}_C$  on the decision-gate labelled by  $x_3$  that there are 2 such solutions. Hence, each solution of the decision-gate labelled by  $x_2$  can be extended into 2 full solutions. Hence, we know that there are 2 solutions where  $x_2 = 0$  and 4 solutions where  $x_2 = 2$ . Hence, similarly as before, we can deduce that  $\tau_{13}(x_2) = 2$ . We find the value of  $x_3$  similarly as before.

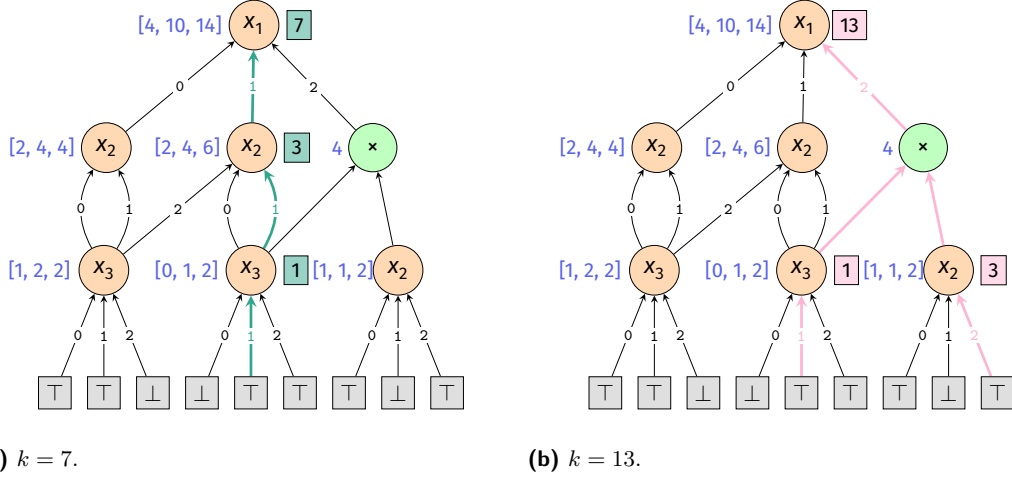
► **Lemma 7** ( $\star$ ). *Assume that we are given a relation  $R \subseteq D^X$  with  $X = \{x_1, \dots, x_n\}$  and an oracle such that for every prefix assignment  $\tau$  of size  $p$  it returns  $\#\sigma_{\tau \wedge x_{p+1} \leq d}(R)$ . Then, for any  $k$ , we can compute  $R[k]$  using  $\mathcal{O}(n \cdot \text{polylog}|D|)$  oracle calls, where  $n = |X|$ .*

**Proof sketch.** Let  $\tau = R[k]$ . We apply Lemma 1 iteratively to find  $\tau(x_i)$  for each  $i = 1, \dots, n$ . By Lemma 1, we know that  $\tau(x_1) = d$  where  $d$  is the smallest value in  $D$  such that  $\#\sigma_{x_1 \leq d}(R) \geq k$ . Now one can easily find  $d$  using a binary search that makes  $\mathcal{O}(\log|D|)$  oracle calls. Now, finding  $\tau(x_2)$  boils down to find  $R'[k']$  where the  $k' = k - \#\sigma_{x_1 < d}(R)$  and  $R' = \sigma_{x_1 = d}(R)$  which can be done similarly using oracle calls for  $R'$ , which could be simulated by oracle calls to  $R$ , and so on for each value  $\tau(x_i)$ . ◀

Now, it remains to show that oracle calls as in Lemma 7 can be efficiently solved on ordered  $\{\times, \text{dec}\}$ -circuits after preprocessing.



## 13:10 Direct Access for Conjunctive Queries with Negations



■ **Figure 3** Examples of paths followed in the circuit for different direct access tasks. Precomputed values of  $\text{nrel}_C$  for decision gates are represented as lists at the left of each gate. The current index of the tuple we are searching for is indicated next to the reached nodes.

► **Lemma 8** ( $\star$ ). *For a given prefix assignment  $\tau$  of size  $p$  and  $d \in D$ , assuming that  $|\text{rel}(v)|$ ,  $\text{nrel}_C(v, d)$  and  $\text{var}(v)$  have been precomputed for every gate  $v$  in  $C$ ,  $\#\sigma_{\tau \wedge x_{p+1} \leq d}(\text{rel}(C))$  can be computed in time  $\mathcal{O}(\text{poly}(n)\text{polylog}|D|)$ , where  $n = |X|$ .*

**Proof sketch.** To ease notation, let  $x = x_{p+1}$ . We first compute  $f_\tau$  in  $\mathcal{O}(n)$  with Lemma 3. Now,  $\sigma_\tau(\text{rel}(C) = D^\Delta \times \text{rel}(f_\tau) \times \{\tau\}$  for  $\Delta = \{x_{p+1}, \dots, x_n\} \setminus \text{var}(f_\tau)$ . Now, either  $x \in \Delta$  and we have  $\sigma_{\tau \wedge x \leq d}(\text{rel}(C)) = D^{\Delta \setminus \{x\}} \times \sigma_{x \leq d}(D^{\{x\}}) \times \text{rel}(f_\tau) \times \{\tau\}$  or  $x_{p+1} \notin \Delta$  and we can show that there exists a decision gate  $v$  in  $f_\tau$  such that  $\text{decvar}(v) = x$ . In this case,  $\sigma_{\tau \wedge x \leq d}(\text{rel}(C)) = D^\Delta \times \sigma_{x \leq d}(\text{rel}(v)) \times \text{rel}(f_\tau \setminus \{v\}) \times \{\tau\}$ . In any case, we can compute each part of the Cartesian product from precomputed values:  $|\text{rel}(f_\tau)| = \prod_{w \in f_\tau} |\text{rel}(w)|$ ,  $|D^\Delta| = |D|^{|\Delta|}$ ,  $\#\sigma_{x \leq d}(D^{\{x\}}) = \text{rank}(d)$  and  $\#\sigma_{x \leq d}(\text{rel}(v)) = \text{nrel}_C(v, d)$ , hence  $\#\sigma_{\tau \wedge x_{p+1} \leq d}(\text{rel}(C))$  is computed with at most  $n$  multiplications of integers of size at most  $n \cdot \text{polylog}(|D|)$ . ◀

Theorem 4 now follows from Lemmas 5, 7, and 8. Indeed, after preprocessing the circuit using Lemma 8, one can use the procedure described in Lemma 7 to solve direct access task using oracle calls, that can be answered efficiently as shown in Lemma 8.

### 4 From join queries to ordered $\{\times, \text{dec}\}$ -circuits

In this section, we present a simple top-down algorithm such that on input  $Q$ ,  $\prec$  and  $\mathbf{D}$ , it returns a  $\succ$ -ordered  $\{\times, \text{dec}\}$ -circuit  $C$  such that  $\text{rel}(C) = Q(\mathbf{D})$ , where  $Q$  is a join query,  $\prec$  an order on its variables and  $\mathbf{D}$  a database. This algorithm is an adaptation of exhaustive DPLL [31], an algorithm that has been originally devised to solve the  $\#\text{SAT}$  problem, but Huang and Darwiche [20] have shown that the trace of this algorithm implicitly builds a Boolean circuit, corresponding to the  $\{\times, \text{dec}\}$ -circuits on domain  $\{0, 1\}$ . We show how to adapt it in the framework of signed join queries. The algorithm itself is presented in Section 4.1. We study the complexity of this algorithm in Section 4.3 depending on the structure of  $Q$  and  $\prec$ , using hypergraph structural parameters introduced in Section 4.2.

#### 4.1 Exhaustive DPLL for signed join queries

The main idea of DPLL for signed join queries is the following: given an order  $\prec$  on the variables of a join query  $Q$  and a database  $\mathbf{D}$ , we construct a  $\succ$ -ordered  $\{\times, \text{dec}\}$ -circuit (where  $x \succ y$  iff  $y \prec x$ )<sup>1</sup> computing  $\llbracket Q \rrbracket^{\mathbf{D}}$  by successively testing the variables of  $Q$  with decision gates, from the highest to the lowest wrt  $\prec$ . At its simplest form, the algorithm picks the highest variable  $x$  of  $Q$  wrt  $\prec$ , creates a new decision gate  $v$  on  $x$  and then, for every value  $d \in D$ , sets  $x$  to  $d$  and recursively computes a gate  $v_d$  computing the subset of  $\llbracket Q \rrbracket^{\mathbf{D}}$  where  $x = d$ . We then add  $v_d$  as an input of  $v$  and proceed with the next value  $d' \in D$ . This approach is however not enough to get interesting tractability results. We hence add the following optimisations. First, we keep a cache of already computed queries so that if we recursively call the algorithm twice on the same input, we can directly return the previously constructed gate. Moreover, if we detect that the answers of  $Q$  are the Cartesian product of two or more subqueries  $Q_1, \dots, Q_k$ , then we create a new  $\times$ -gate  $v$ , recursively call the algorithm on each component  $Q_i$  to construct a gate  $w_i$  and plug each  $w_i$  to  $v$ . Detecting such cases is mainly done syntactically, by checking whether the query can be partitioned into subqueries having disjoint variables. However, this approach would fail to give good complexity bounds in the presence of negative atoms. To achieve the best complexity, we also remove from  $Q$  every negative atom as soon as it is satisfied by the current partial assignment. This allows us to discover more cases where the query has connected components. The theoretical performance of the previously described algorithm may however vary if one is not careful in the way the recursive calls are actually made. We hence give a more formal presentation of the algorithm, whose pseudocode is presented in Algorithm 1.

■ **Algorithm 1** An algorithm to compute a  $\succ$ -ordered  $\{\times, \text{dec}\}$ -circuit representing  $\llbracket Q \rrbracket^{\mathbf{D}}$ .

---

```

1: procedure DPLL( $Q, \tau, \mathbf{D}, \prec$ )
2:   if ( $Q, \tau$ ) is in cache then return cache( $Q, \tau$ )
3:   if  $Q$  is inconsistent with  $\tau$  then return  $\perp$ -gate
4:   if  $\tau$  assigns every variable in  $Q$  then return  $\top$ -gate
5:    $x \leftarrow \max_{\prec} \text{var}(Q)$ 
6:   for  $d \in D$  do
7:      $\tau' \leftarrow \tau \times [x \leftarrow d]$ 
8:     if  $Q$  is inconsistent with  $\tau'$  then  $v_d \leftarrow \perp$ -gate
9:     else
10:      Let  $Q_1, \dots, Q_k$  be the  $\tau'$ -connected components of  $Q \downarrow \tau'$ 
11:      for  $i = 1$  to  $k$  do  $w_i \leftarrow \text{DPLL}(Q_i, \tau_i, \mathbf{D}, \prec)$  where  $\tau_i = \tau' \upharpoonright_{\text{var}(Q_i)}$ 
12:       $v_d \leftarrow \text{new } \times$ -gate with inputs  $w_1, \dots, w_k$ 
13:     end if
14:   end for
15:    $v \leftarrow \text{new dec-gate}$  connected to  $v_d$  by a  $d$ -labelled edge for every  $d \in D$ 
16:   cache( $Q, \tau$ )  $\leftarrow v$ 
17:   return  $v$ 
18: end procedure

```

---

<sup>1</sup> While one could easily change the algorithm so that it produces a  $\prec$ -ordered  $\{\times, \text{dec}\}$ -circuit instead, the structural parameters we will be considering for the tractability of DPLL in Section 4.2 are more naturally defined on  $\prec$ . We choose to present DPLL this way to ease the proofs later.

## 13:12 Direct Access for Conjunctive Queries with Negations

A few notations are used in Algorithm 1. Given a database  $\mathbf{D}$  on domain  $D$  and a tuple  $\tau \in D^Y$ , we denote by  $\llbracket Q \rrbracket_{\tau}^{\mathbf{D}}$  the set of tuples  $\sigma \in D^{\text{var}(Q) \setminus Y}$  that are answers of  $Q$  when extended with  $\tau$ . More formally,  $\sigma \in \llbracket Q \rrbracket_{\tau}^{\mathbf{D}}$  if and only if  $(\sigma \times \tau)|_{\text{var}(Q)} \in \llbracket Q \rrbracket^{\mathbf{D}}$ . Given an atom  $R(\mathbf{x})$ , a database  $\mathbf{D}$  and a tuple  $\tau \in D^Y$ , we say that  $R(\mathbf{x})$  is *inconsistent with  $\tau$  wrt  $\mathbf{D}$*  (or simply inconsistent with  $\tau$  when  $\mathbf{D}$  is clear from context) if there is no  $\sigma \in R^{\mathbf{D}}$  such that  $\tau \simeq \sigma$ . Observe that if  $Q$  contains a positive atom  $R(\mathbf{x})$  that is inconsistent with  $\tau$  then  $\llbracket Q \rrbracket_{\tau}^{\mathbf{D}} = \emptyset$ . Similarly, if  $Q$  contains a negative atom  $\neg R(\mathbf{x})$  such that  $\tau$  assigns every variable of  $\mathbf{x}$  and  $\tau(\mathbf{x}) \in R$ , then  $\llbracket Q \rrbracket_{\tau}^{\mathbf{D}} = \emptyset$ . If one of these cases arises, we say that  $Q$  is *inconsistent with  $\tau$* . Now observe that if  $\neg R(\mathbf{x})$  is a negative atom of  $Q$  such that  $R(\mathbf{x})$  is inconsistent with  $\tau$ , then  $\llbracket Q \rrbracket_{\tau}^{\mathbf{D}} = \llbracket Q' \rrbracket_{\tau}^{\mathbf{D}} \times D^W$  where  $Q' = Q \setminus \{\neg R(\mathbf{x})\}$  and  $W = \text{var}(Q) \setminus \text{var}(Q')$  (some variables of  $Q$  may only appear in the atom  $\neg R(\mathbf{x})$ ). This motivates the following definition: the *simplification of  $Q$  wrt to  $\tau$  and  $\mathbf{D}$* , denoted by  $Q \Downarrow \langle \tau, \mathbf{D} \rangle$  or simply by  $Q \Downarrow \tau$  when  $\mathbf{D}$  is clear from context, is defined to be the subquery of  $Q$  obtained by removing from  $Q$  every negative atom  $\neg R(\mathbf{x})$  of  $Q$  such that  $R(\mathbf{x})$  is inconsistent with  $\tau$ . From what precedes, we clearly have  $\llbracket Q \rrbracket_{\tau}^{\mathbf{D}} = \llbracket Q' \rrbracket_{\tau}^{\mathbf{D}} \times D^W$  where  $Q' = Q \Downarrow \langle \tau, \mathbf{D} \rangle$  and  $W = \text{var}(Q) \setminus \text{var}(Q')$ .

For a tuple  $\tau \in D^Y$  assigning a subset  $Y$  of variables of  $Q$ , the  $\tau$ -*intersection graph*  $\mathcal{I}_{\tau}^Q$  of  $Q$  is the graph whose vertices are the atoms of  $Q$  having at least one variable not in  $Y$  and there is an edge between two atoms  $a, b$  of  $Q$  if  $a$  and  $b$  share a variable that is not in  $Y$ . Observe that  $\mathcal{I}_{\tau}^Q$  does not depend on the values of  $\tau$  but only on the variables it sets. Hence it can be computed in polynomial time in the size of  $Q$  only. A connected component  $C$  of  $\mathcal{I}_{\tau}^Q$  naturally induces a subquery  $Q_C$  of  $Q$  and is called a  $\tau$ -*connected component*.  $Q$  is partitioned into its  $\tau$ -connected components and the atoms whose variables are completely set by  $\tau$ . More precisely,  $Q = \bigcup_{C \in \mathcal{CC}} Q_C \cup Q'$  where  $\mathcal{CC}$  are the connected component of  $\mathcal{I}_{\tau}^Q$  and  $Q'$  contains every atom  $a$  of  $Q$  on variables  $\mathbf{x}$  such that  $\mathbf{x}$  only has variables in  $Y$ . Observe that if  $\tau$  is an answer of  $Q'$ , then  $\llbracket Q \rrbracket_{\tau}^{\mathbf{D}} = \times_{C \in \mathcal{CC}} \llbracket Q_C \rrbracket_{\tau_C}^{\mathbf{D}}$  where  $\tau_C = \tau|_{\text{var}(Q_C)}$  since if  $C_1$  and  $C_2$  are two distinct  $\tau$ -connected components of  $\mathcal{I}_{\tau}^Q$ , then  $\text{var}(Q_{C_1}) \cap \text{var}(Q_{C_2}) \subseteq Y$ .

► **Example 9.** We illustrate the previous definitions on the signed join query  $Q(x_1, \dots, x_5)$  defined as  $\neg R(x_1, \dots, x_5), S(x_1, x_2, x_3), T(x_1, x_4, x_5)$  and database  $\mathbf{D}$  on domain  $\{0, 1\}$  with  $R^{\mathbf{D}} = \{(1, 1, 1, 1, 1)\}$ . Let  $\tau = [x_1 \leftarrow 0]$ . The  $\tau$ -intersection graph of  $Q$  is a path where  $\neg R(x_1, \dots, x_5)$  is connected to  $S(x_1, x_2, x_3)$  and  $T(x_1, x_4, x_5)$ . There is no edge between  $S(x_1, x_2, x_3)$  and  $T(x_1, x_4, x_5)$  since  $x_1$  is their only common variable and it is assigned by  $\tau$ . Hence,  $Q$  has one  $\tau$ -connected component containing every atom of  $Q$ . Now,  $Q \Downarrow \tau = S(x_1, x_2, x_3), T(x_1, x_4, x_5)$  since  $R(0, x_2, \dots, x_5)$  is inconsistent over  $\mathbf{D}$  and the  $\tau$ -intersection graph of  $Q \Downarrow \tau$  consists in two isolated vertices  $S(x_1, x_2, x_3)$  and  $T(x_1, x_4, x_5)$ . Hence  $Q \Downarrow \tau$  has two  $\tau$ -connected components. This example also illustrates the role of simplification for discovering Cartesian products.

The correctness can be proven by induction: a recursive call  $\text{DPLL}(Q, \tau, \mathbf{D}, \prec)$  returns a gate computing  $\llbracket Q \rrbracket_{\tau}^{\mathbf{D}}$ . This is formalised in the following theorem. We analyse the complexity of DPLL in Section 4.3.

► **Theorem 10** ( $\star$ ). *Let  $Q$  be a signed join query,  $\mathbf{D}$  a database and  $\prec$  an order on  $\text{var}(Q)$ , then  $\text{DPLL}(Q, \langle \cdot \rangle, \mathbf{D}, \prec)$  constructs a  $\succ$ -ordered  $\{\times, \text{dec}\}$ -circuit  $C$  and returns a gate  $v$  of  $C$  such that  $\text{rel}(v) = \llbracket Q \rrbracket^{\mathbf{D}}$ .*

## 4.2 Hyperorder width

In this section, we introduce the notions of width based on elimination orders rather than tree decompositions that are relevant to pinpoint the complexity of the DPLL procedure.

**Order based widths ( $\text{how}(\cdot)$ ,  $\text{fhow}(\cdot)$ ).** A hypergraph  $H = (V, E)$  and an order  $\prec$  such that  $V = \{v_1, \dots, v_n\}$  with  $v_1 \prec \dots \prec v_n$  induces a series of hypergraphs defined as  $H_1^\prec, \dots, H_{n+1}^\prec$  as  $H_1^\prec = H$  and  $H_{i+1}^\prec = H_i^\prec / v_i$ . The *hyperorder width*  $\text{how}(H, \prec)$  of  $\prec$  wrt  $H$  is defined as  $\max_{i \leq n} \text{cn}(N_{H_i^\prec}(v_i), E)$ . The *hyperorder width*  $\text{how}(H)$  of  $H$  is defined as the best possible width using any elimination order, that is,  $\text{how}(H) = \min_{\prec} \text{how}(H, \prec)$ . We similarly define the *fractional hyperorder width*  $\text{fhow}(H, \prec)$  of  $\prec$  wrt  $H$  as  $\max_{i \leq n} \text{fcn}(N_{H_i^\prec}(v_i), E)$  and the *fractional hyperorder width*  $\text{fhow}(H)$  of  $H$  as  $\text{fhow}(H) = \min_{\prec} \text{fhow}(H, \prec)$ .

It has already been observed many times ([23, Appendix C] or [16, 17, 24]) that  $\text{how}(H)$  and  $\text{fhow}(H)$  are respectively equal to the generalised hypertree width and the fractional hypertree width of  $H$  and that there is a natural correspondence between a tree decomposition and an elimination order having the same width. However, to be able to express our tractability results as function of the order, it is more practical to define the width of orders instead of hypertree decompositions. In [7, Definition 9],  $\text{fhow}(H, \prec)$  is called the incompatibility number, though it is not formally defined on hypergraphs but directly on conjunctive queries. The case  $k = 1$ , which corresponds to the  $\alpha$ -acyclicity of the underlying hypergraph, has also been previously called an order without disruptive trio [12]. However, these notions are specifically used for the problem of direct access in conjunctive queries while the characterisation of hypergraph measures in terms of elimination orders of hypergraphs predates by several years this terminology (see [5] for a survey). We then choose a terminology closer to the usual terminology for hypergraph decompositions.

**Hereditary order based widths ( $\beta\text{-how}(\cdot)$ ,  $\beta\text{-fhow}(\cdot)$ ).** Hypertree width is not hereditary. That is, the (fractional) hypertree width of a subhypergraph can be much bigger than the (fractional) hypertree width of the hypergraph itself. It makes it not well suited to discover tractable classes for signed join queries. Indeed, if a query  $Q$  contains a negative atom  $\neg R(\mathbf{x})$  and if  $R^{\mathbf{D}}$  is empty in the database  $\mathbf{D}$ , then  $\llbracket Q \rrbracket^{\mathbf{D}}$  is equal to  $\llbracket Q' \rrbracket^{\mathbf{D}}$ , where  $Q' = Q \setminus \{\neg R(\mathbf{x})\}$ . Hence if some aggregation problem for a fixed self-join free query  $Q$  on an input database  $\mathbf{D}$  can be solved in  $O(\text{poly}(|\mathbf{D}|))$  for any database  $\mathbf{D}$ , it has to be tractable for every  $Q'$  obtained by removing a subset of the negative atoms from  $Q$ . This motivates the following definitions: for a hypergraph  $H = (V, E)$  and an order  $\prec$  on  $V$ , the  $\beta$ -hyperorder width  $\beta\text{-how}(H, \prec)$  of  $\prec$  wrt to  $H$  is defined as  $\max_{H' \subseteq H} \text{how}(H', \prec)$ . The  $\beta$ -hyperorder width  $\beta\text{-how}(H)$  of  $H$  is defined as the width of the best possible elimination order, that is,  $\beta\text{-how}(H) = \min_{\prec} \beta\text{-how}(H, \prec)$ . We define similarly the  $\beta$ -fractional hyperorder width of an order  $\prec$  and of an hypergraph –  $\beta\text{-fhow}(H, \prec)$  and  $\beta\text{-fhow}(H)$  – by replacing  $\text{how}(\cdot)$  by  $\text{fhow}(\cdot)$  in the definitions.

**Comparison with existing measures.** The fact that fractional hypertree width is not hereditary has traditionally been worked around by taking the largest width over every subhypergraph. In other words, the  $\beta$ -fractional hypertree width  $\beta\text{-fhtw}(H)$  of  $H$  is defined as  $\beta\text{-fhtw}(H) = \max_{H' \subseteq H} \text{fhtw}(H')$ . The  $\beta$ -hypertree width  $\beta\text{-htw}(H)$  is defined similarly. If one plugs the ordered characterisation of  $\text{fhtw}(H')$  in this definition, one can observe that  $\beta\text{-fhtw}(H) = \max_{H' \subseteq H} \min_{\prec} \text{fhow}(H', \prec)$ . Hence, the difference between  $\beta\text{-fhtw}(H)$  and  $\beta\text{-fhow}(H)$  boils down to inverting the min and the max in the definition. It directly gives that  $\beta\text{-fhtw}(H) \leq \beta\text{-fhow}(H)$  and  $\beta\text{-htw}(H) \leq \beta\text{-how}(H)$  for every  $H$ . The main advantage of the  $\beta$ -fractional hyperorder width is that it comes with a natural notion of decomposition – the best elimination order  $\prec$  – that can be used algorithmically. This is not given by the definition of  $\beta\text{-fhtw}(\cdot)$  and has yet to be found.

The only exception is the case where  $\beta\text{-fhtw}(H) = 1$ , known as  $\beta$ -acyclicity, where an order-based characterisation is known and has been used to show the tractability of many problems such as SAT [29], #SAT or #CQ for  $\beta$ -acyclic instances [9, 6]. The elimination order is based on the notion of nest points. In a hypergraph  $H = (V, E)$ , a *nest point* is a vertex  $v \in V$  such that  $E(v)$  is ordered by inclusion, that is,  $E(v) = \{e_1, \dots, e_p\}$  with  $e_1 \subseteq \dots \subseteq e_p$ . A  $\beta$ -*elimination order*  $(v_1, \dots, v_n)$  for  $H$  is an ordering of  $V$  such that for every  $i \leq n$ ,  $v_i$  is a nest point of  $H \setminus \{v_1, \dots, v_{i-1}\}$ . A closer inspection of the definition of  $\beta$ -elimination order  $\prec$  shows that  $\beta\text{-fhtw}(H, \prec) = \beta\text{-how}(H, \prec) = 1$ , showing that it corresponds to  $\beta$ -acyclicity. We can actually prove a more general result: the notion of  $\beta$ -acyclicity has been recently generalised by Lanzinger in [25] using a notion called nest sets. A set of vertices  $S \subseteq V$  is a *nest set of  $H$*  if  $\{e \setminus S \mid e \in E, e \cap S \neq \emptyset\}$  is ordered by inclusion. A *nest set elimination order* is a list  $\Pi = (S_1, \dots, S_p)$  such that:  $\bigcup_{i=1}^p S_i = V$ ,  $S_i \cap S_j = \emptyset$  and  $S_i$  is a nest set of  $H \setminus \bigcup_{j < i} S_j$ .

The width of a nest set elimination is  $\text{nsw}(H, \Pi) = \max_i |S_i|$  and the *nest set width*  $\text{nsw}(H)$  of  $H$  is defined to be the smallest possible width of a nest set elimination order of  $H$ . It turns out that our notion of width generalises the notion of nest set width, that is, we have  $\beta\text{-how}(H) \leq \text{nsw}(H)$ . More particularly, any order  $\prec$  obtained from a nest set elimination order  $\Pi = (S_1, \dots, S_p)$  by ordering each  $S_i$  arbitrarily verifies  $\text{nsw}(H, \Pi) \geq \beta\text{-how}(H, \prec)$ .

We summarise the above discussion in the following theorem:

► **Theorem 11** ( $\star$ ). *For every hypergraph  $H = (V, E)$ , we have:  $\beta\text{-htw}(H) \leq \beta\text{-how}(H) \leq \text{nsw}(H)$ . In particular,  $H$  is  $\beta$ -acyclic iff  $\beta\text{-how}(H) = 1$ .*

The goal of this paper is not to give a thorough analysis of  $\beta$ -fractional hyperorder width so we leave for future research several questions related to it: what is the complexity of computing the  $\beta$ -fractional hyperorder width of a hypergraph, how does it compare with other widths such as (incidence) treewidth, (incidence) cliquewidth, MIM-width or point-width [11]. For these measures of width, #SAT, a problem close to computing the number of answers in signed join queries, is known to be tractable (see [9] for a survey). We leave open the most fundamental question of comparing the respective powers of  $\beta\text{-fhtw}(\cdot)$  and  $\beta\text{-how}(\cdot)$ . We also refer the interested reader to the full version where we discuss why the seemingly natural notion of  $\beta$ -hyperorder width has not appeared earlier in the literature.

**Signed hyperorder width.** In the case of signed join queries, one can deal with positive and negative atoms differently, which is not reflected by the definition of  $\beta\text{-how}(\cdot)$ . We generalise these widths to signed hypergraphs by taking subhypergraphs only on the negative part, generalising a notion of acyclicity introduced by Brault-Baron in [4] that mixes  $\beta$ - and  $\alpha$ -acyclicities for signed hypergraphs. Let  $H = (V, E_+, E_-)$  be a signed hypergraph. Given an order  $\prec$  on  $V$ , the *signed hyperorder width*  $\text{show}(H, \prec)$  of  $\prec$  wrt  $H$  is defined as  $\text{show}(H, \prec) = \max_{E' \subseteq E_-} \text{how}((V, E_+ \cup E'), \prec)$ . The *signed hyperorder width*  $\text{show}(H)$  of  $H$  is defined as  $\text{show}(H) = \min_{\prec} \text{show}(H, \prec)$ . Fractional version of these widths could easily be defined but will not be needed in this paper. It is clear from the definition that if  $E_+ = \emptyset$  then  $\text{show}(H, \prec) = \beta\text{-how}(H, \prec)$  and if  $E_- = \emptyset$ , then  $\text{show}(H, \prec) = \text{how}(H, \prec)$ .

### 4.3 Complexity of exhaustive DPLL

The complexity of DPLL on a conjunctive query  $Q$  and order  $\prec$  can be bounded in terms of the hyperorder width of  $H(Q)$  wrt  $\prec$ :

► **Theorem 12.** *Let  $Q$  be a signed join query,  $\mathbf{D}$  a database over domain  $D$  and  $\prec$  an order on  $\text{var}(Q)$ . Then  $\text{DPLL}(Q, \langle \rangle, \mathbf{D}, \prec)$  produces a  $\succ$ -ordered  $\{\times, \text{dec}\}$ -circuit  $C$  of size  $\mathcal{O}(\text{poly}_k(|Q|)|\mathbf{D}|^{k+1})$  in time  $\mathcal{O}(\text{poly}_k(|Q|)|\mathbf{D}|^{k+1}\text{polylog}|D|)$  such that  $\text{rel}(C) = \llbracket Q \rrbracket^{\mathbf{D}}$  where  $k = \text{fhow}(H(Q), \prec)$  if  $Q$  is positive and  $k = \text{show}(H(Q), \prec)$  otherwise.*

A full proof of Theorem 12 can be found in the full version. The proof is technical since there is nothing connecting the structure of the hypergraph of  $Q$  and the runtime of DPLL. Due to space constraints, we only give a short overview of the proof, trying to stress how both notions connect.

In this section, we fix a signed join query  $Q$  that has exactly one  $\langle \rangle$ -component (the case where  $Q$  has many  $\langle \rangle$ -component can be easily dealt with by constructing the Cartesian product of each  $\langle \rangle$ -component of  $Q$ ), a database  $\mathbf{D}$  and an order  $\prec$  on  $\text{var}(Q) = \{x_1, \dots, x_n\}$  where  $x_1 \prec \dots \prec x_n$ . We let  $D$  be the domain of  $\mathbf{D}$ ,  $n$  be the number of variables of  $Q$  and  $m$  be the number of atoms of  $Q$ . To ease notation, we will write  $X$  instead of  $\text{var}(Q)$ . For  $i \leq n$ , we denote  $\{x_1, \dots, x_i\}$  by  $X_{\leq i}$ . Similarly,  $X_{< i} = X_{\leq i} \setminus \{x_i\}$ ,  $X_{> i} = \text{var}(Q) \setminus X_{\leq i}$  and  $X_{\geq i} = \text{var}(Q) \setminus X_{< i}$ . Finally, we let  $\mathbf{R}_Q^{\mathbf{D}}$  be the set of  $(K, \sigma)$  such that  $\text{DPLL}(Q, \langle \rangle, \mathbf{D}, \prec)$  makes at least one recursive call to  $\text{DPLL}(K, \sigma, \mathbf{D}, \prec)$ . The first thing to observe, is that thanks to the usage of a cache, the complexity of DPLL can naturally be stated as a function of  $|\mathbf{R}_Q^{\mathbf{D}}|$ :

► **Lemma 13** ( $\star$ ).  *$\text{DPLL}(Q, \langle \rangle, \mathbf{D}, \prec)$  produces a circuit of size at most  $\mathcal{O}(|\mathbf{R}_Q^{\mathbf{D}}| \cdot |D| \cdot \text{poly}(|Q|))$  in time  $\mathcal{O}(|\mathbf{R}_Q^{\mathbf{D}}| \cdot \text{poly}(|Q|) \cdot |D| \text{polylog}|D|)$ .*

Hence to prove Theorem 12, one only needs to bound the size of  $\mathbf{R}_Q^{\mathbf{D}}$ . To do that, we characterise the elements  $(K, \sigma) \in \mathbf{R}_Q^{\mathbf{D}}$  in terms of hypergraph structure. The key notion we need is the notion of  $x$ -components, a definition akin to the one used in [9] to compile  $\beta$ -acyclic CNF formulas. Let  $Q' \subseteq Q$  be a subquery of  $Q$  and  $x, y$  two variables of  $Q'$  such that  $y \prec x$ . An  $x$ -path to  $y$  in  $Q'$  is a list  $x_0, a_0, \dots, x_p$  where  $a_i \in \text{atoms}(Q')$  is an atom of  $Q'$  on variables  $\mathbf{x}_i$ ,  $x_i$  is a variable of  $\mathbf{x}_i$ ,  $x_0 = x$ ,  $x_p = y$  and  $x_i \preceq x$  for every  $i \leq p$ . Intuitively, it maps to a path in the hypergraph of  $Q'$  that starts from  $x$  and is only allowed to use vertices smaller than  $x$ . The  $x$ -component of  $Q'$  is the set of atoms  $a$  of  $Q'$  such that there exists an  $x$ -path to a variable  $y$  of  $a$  in  $Q'$ .

It turns out that the recursive calls of DPLL are  $x$ -components of some  $Q' \subseteq Q$  and  $x \in X$  where  $Q'$  is obtained from  $Q$  by removing negative atoms. Intuitively, these removed atoms are the ones that cannot be satisfied anymore by the current assignment of variables.

► **Lemma 14** ( $\star$ ). *Let  $(K, \sigma) \in \mathbf{R}_Q^{\mathbf{D}}$  and let  $x$  be the biggest variable of  $K$  not assigned by  $\sigma$ . There exists  $\tau$  a partial assignment of  $X_{> x}$  such that  $\tau|_{\text{var}(K)} = \sigma$  and  $K$  is the  $x$ -component of  $Q \Downarrow \tau$ .*

Moreover,  $x$ -components are connected to the width notions from Section 4.2 as follows:

► **Lemma 15** ( $\star$ ). *Let  $Q$  be a signed join query on variables  $X = \{x_1, \dots, x_n\}$ ,  $x_i$  a variable of  $Q$  and  $K_i$  its  $x_i$ -component. We let  $H$  be the hypergraph of  $Q$ ,  $H_1 = H$  and  $H_{j+1} = H_j/x_j$ . We have  $N_{x_i}(H_i) = \text{var}(K_i) \cap X_{\geq x_i}$ .*

Intuitively, Lemma 15 allows us to conclude that the variables from  $X_{> x}$  of an  $x$ -component in a hypergraph  $H$  admit a fractional cover of value at most  $k$  where  $k = \text{fhow}(H, \prec)$ . Now we illustrate how one can use it to get a bound on  $\mathbf{R}_Q^{\mathbf{D}}$ . We start with the case of positive conjunctive query. Let  $(K, \sigma) \in \mathbf{R}_Q^{\mathbf{D}}$ ,  $\sigma$ . By Lemma 14 that there exists  $\tau \supset \sigma$  such that  $K$  is the  $x$ -component of  $Q \Downarrow \tau$ , which is equal to  $Q$  in the case of positive conjunctive query. Moreover,  $\tau$  must be compatible with every atom of  $Q$ , otherwise DPLL would have returned



## 13:16 Direct Access for Conjunctive Queries with Negations

$\perp$  already. In other words,  $\tau$  satisfies every atom from the  $x$ -component of  $Q$  restricted to  $X_{\succ x}$ . But these atoms have a fractional cover of value at most  $k$ , hence by Theorem 2, there are at most  $|\mathbf{D}|^k$  different  $\tau$  for each  $x$ -component. Since there are at most  $n$  distinct  $x$ -component, a bound on  $\mathbf{R}_Q^{\mathbf{D}}$  follows.

We can have a similar bound for the signed case but now  $Q \Downarrow \tau$  is not  $Q$  anymore since some negative atoms may have been removed from  $Q$  because they are incompatible with  $\tau$ . However, we know that the hypergraph of  $Q \Downarrow \tau$  is obtained by removing negative atoms of  $Q$ , hence we know that the variables from  $X_{\succ x}$  of any  $x$ -component of  $Q \Downarrow \tau$  are covered by at most  $k$  atoms. Moreover,  $Q \Downarrow \tau$  only contains atoms that are compatible with  $\tau$ . Hence by Theorem 2 again, we can show that for a given  $K$ , there are at most  $|\mathbf{D}|^k$  distinct  $\tau$  such that  $(K, \tau) \in \mathbf{R}_Q^{\mathbf{D}}$ . Moreover, we can show that the possible  $K$  can be characterised by looking at the  $x$ -component of  $Q \Downarrow \tau$  for every  $\tau$  that is a solution of the join of at most  $k$  atoms projected on  $X_{\succ x}$  which allows us to derive an  $m^{k+1}|\mathbf{D}|^k$  bounds on  $\mathbf{R}_Q^{\mathbf{D}}$  in this case. To wrap up, we can prove the following bounds on  $\mathbf{R}_Q^{\mathbf{D}}$ :

► **Lemma 16** ( $\star$ ). *If  $Q$  is a positive join query with  $m$  atoms and  $n$  variables,  $|\mathbf{R}_Q^{\mathbf{D}}| \leq n|\mathbf{D}|^k$  where  $k = \text{fhw}(H(Q), \prec)$ . Otherwise  $|\mathbf{R}_Q^{\mathbf{D}}| \leq nm^{k+1}|\mathbf{D}|^k$  where  $k = \text{show}(H(Q), \prec)$ .*

Theorem 12 is a direct corollary of Lemmas 13 and 16. One may wonder why we do not use fractional width when  $Q$  contains negative atoms. The proof of Lemma 16 breaks in this case when we try to bound the number, for a given  $x$ , of  $x$ -component  $K$  that can appear in recursive calls. To prove Lemma 16, we bound it by taking at subset of at most  $k$  atoms of  $Q$ . To do it with fractional cover, one would need to consider every combination of atoms of  $Q$  having fractional cover at most  $k$  which we did not manage to bound by a polynomial in  $Q$ . We therefore leave this question open for future research but observe that it would give a complexity of at most  $\mathcal{O}(2^m |\mathbf{D}|^{k+1} \text{polylog}|D|)$  which is polynomial wrt data complexity.

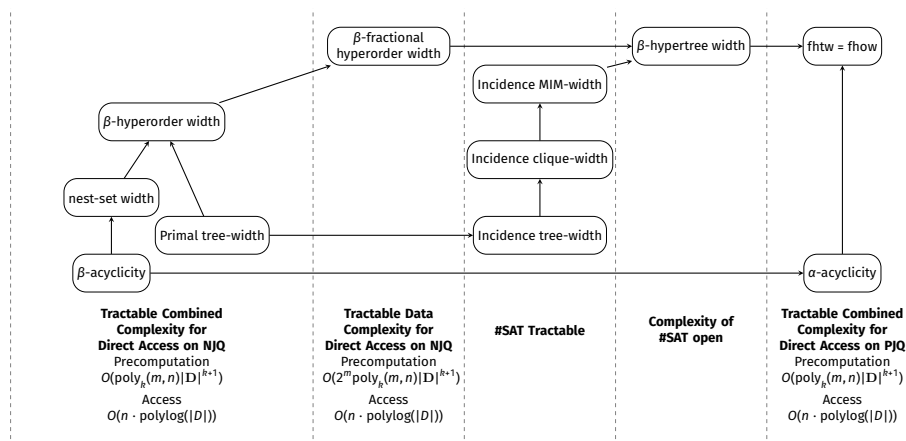
## 5 Tractability results for signed join and conjunctive queries

Combining the algorithm for DA on ordered circuits from Section 3 with the algorithm of Section 4 gives tractability results on the complexity of direct access for signed join queries:

► **Theorem 17** ( $\star$ ). *Given a signed join query  $Q$ , an order  $\prec$  on  $\text{var}(Q)$  and a database  $\mathbf{D}$  on domain  $D$ , we can solve the direct access problem for  $\prec_{\text{lex}}$  with precomputation  $\mathcal{O}(|\mathbf{D}|^{k+1} \text{polylog}|D| \cdot \text{poly}_k(|Q|))$  and access time  $\mathcal{O}(\text{poly}(n) \cdot \text{polylog}|D|)$  where  $n = |\text{var}(Q)|$  where  $k = \text{fhtw}(H(Q), \succ)$  if  $Q$  is positive and  $k = \text{show}(H(Q), \succ)$  otherwise.*

**Negative join queries and #SAT.** Theorem 17 generalises many tractability results from the literature. First of all, our result can directly be applied to #SAT, the problem of counting the number of satisfying assignment of a CNF formula. A CNF formula  $F$  with  $m$  clauses can directly be transformed into a negative join query  $Q_F$  with  $m$  atoms having the same hypergraph and into a database  $\mathbf{D}_F$  on domain  $\{0, 1\}$  and of size at most  $m$  such that  $[[Q_F]]^{\mathbf{D}_F}$  is the set of satisfying assignments of  $F$ . Indeed, a clause can be seen as the negation of a relation having exactly one tuple. For example,  $x \vee y \vee \neg z$  can be seen as  $\neg R(x, y, z)$  where  $R$  contains the tuple  $(0, 0, 1)$ . Hence, Theorem 17 generalises both [9] and [6] by providing a compilation algorithm for  $\beta$ -acyclic queries to any domain size and to the more general measure of  $\beta$ -hyperorder width. It also shows that not only counting is tractable but also the more general DA tasks. Theorem 17 also generalises the results of [25] which shows the tractability of the evaluation of negative join queries with bounded nest set width. Since a negative join query with nest set width  $k$  has  $\beta$ -hyperorder width at most





■ **Figure 4** Landscape of hypergraph measures and known inclusions (depicted as an arrow) with tractability results for direct access on positive and negative join queries (PJQ/NJQ) and #SAT on CNF formulas. Here  $n$  is the number of variables,  $\mathbf{D}$  the database,  $D$  the domain ( $\{0, 1\}$  for #SAT),  $m$  the number of atoms/clauses and  $k$  the width measure ( $k = 1$  for  $\alpha$ - and  $\beta$ -acyclicity).

$k$  by Theorem 11, Theorem 17 implies that DA is tractable for the class of queries with bounded nest set width. In particular, counting the number of answers is tractable for this class, a question left open in [25].

Figure 4 summarises our contributions for join queries with negations and locates them in the landscape of known tractability results. Even if our result applies to signed conjunctive query, we summarise our contribution only for negative join queries and positive join queries since it allows to compare hypergraph measures (where tractability of signed queries is stated using signed hypergraphs parameters). The stated complexity are given assuming that a decomposition is provided in the input. While the complexity of computing the  $\beta$ -fractional hyperorder width is open, we observe that for nest set width, this decomposition could also be computed in FPT time in the size of the hypergraph [25]. In particular, it gives the following:

► **Theorem 18.** *Computing  $\#[Q]^{\mathbf{D}}$  can be computed in polynomial time when parametrized by nest-set width.*

Despite the fact that computing an optimal nest-set width elimination order is FPT, Theorem 18 only gives an XP algorithm for #SAT and not an FPT algorithm since the complexity of DPLL has a  $O(m^k)$  dependency where  $m$  is the number of atoms.

**Direct access for positive conjunctive queries.** Theorem 17 allows to recover the tractability of DA for positive join queries with bounded fractional hypertree width proven in [12, 7]. Indeed, given an order  $\prec$  on the vertices of a hypergraph, [7] introduces the notion of incompatibility number of  $\prec$  which corresponds exactly to its fractional hyperorder width. Hence Theorem 17 implies the same tractability results for positive join query as [7, Theorem 10]. The complexity bounds from this paper are however better than ours and proven optimal since the preprocessing is of the form  $\text{poly}_k(Q)|\mathbf{D}|^k$  where we have  $\text{poly}_k(Q)|\mathbf{D}|^{k+1}$ . We nevertheless believe that with a more careful analysis of the implementation of Algorithm 1, we could match this upper bound although this is not the focus of this paper. Another strong point of [12] (and also [8, Theorem 39] which is the arXiv version of [7]) is that it handles conjunctive queries, that is, join queries with projection which is not covered by Theorem 17. We demonstrate the versatility of the circuit-based approach by showing how one can also handle quantifiers directly on the circuit.

► **Theorem 19** (★). *Let  $C$  be a  $\prec$ -order circuit on domain  $D$ , variables  $X = \{x_1, \dots, x_n\}$  such that  $x_1 \prec \dots \prec x_n$  and  $j \leq n$ . One can compute in time  $O(|C| \cdot \text{poly}(n) \cdot \text{polylog}(|D|))$  a circuit  $C'$  of size at most  $|C|$  such that  $\text{rel}(C') = \text{rel}(C)|_{\{x_1, \dots, x_j\}}$ .*

Now we can use Theorem 19 to handle conjunctive queries by first using Theorem 12 on the underlying join query to obtain a  $\prec$ -circuit and then by projecting the variables directly in the circuit. This approach works only when the largest variables in the circuits are the quantified variables. It motivates the following definition: given a hypergraph  $H = (V, E)$ , an elimination order  $(v_1, \dots, v_n)$  of  $V$  is  $S$ -connex if and only if there exists  $i$  such that  $\{v_i, \dots, v_n\} = S$ . In other words, the elimination order starts by eliminating  $V \setminus S$  and then proceeds to  $S$ . Given a conjunctive query  $Q$  and an elimination order  $\prec$  on  $\text{var}(Q)$ , we say that the elimination is free-connex if it is a  $\text{free}(Q)$ -connex elimination order of  $H(Q)$  where  $\text{free}(Q)$  are the free variables of  $Q$ <sup>2</sup>. We directly have the following:

► **Theorem 20** (★). *Given a conjunctive query  $Q(Y)$ , a free-connex order  $\succ$  on  $\text{var}(Q)$  and a database  $\mathbf{D}$  on domain  $D$ , we can solve direct access tasks for  $\prec_{\text{lex}}$  with precomputation  $\mathcal{O}(|\mathbf{D}|^{k+1} \text{polylog}|D| \cdot \text{poly}_k(|Q|))$  and access time  $\mathcal{O}(n \cdot \text{polylog}(|D|))$  where  $n = |\text{var}(Q)|$  where  $k = \text{fhtw}(H(Q), \succ)$  if  $Q$  is positive and  $k = \text{show}(H(Q), \succ)$  otherwise.*

We observe that our notion of free-connex elimination order for  $Q$  is akin to [8, Definition 38] with two differences: first, in [8], it is allowed to only specify a preorder on  $\text{free}(Q)$  and the complexity of the algorithm is then stated with the best possible compatible ordering, which would be possible in our framework too. The second difference is that the orders are presented in reverse, that is, in their definition, the orders start with free variables and end with quantified variables. We decided to present free-connexity of elimination orders in this way so that it corresponds to the existing notion of free-connexity of tree decompositions. Now, Theorem 20 constructs a direct access for  $\prec_{\text{lex}}$  when  $\succ$  is free-connex, so Theorem 20 proves the same tractability result as [8, Theorem 39], again with an extra  $|D|$  factor but compatible with negative and signed conjunctive queries.

## 6 Future Work

Our new tractability results for solving DA tasks on signed conjunctive queries relies on a unifying framework for both positive and signed queries using factorised representation of the answer sets of the query. It opens many avenues for research. First, contrary to the positive query case, we do not yet have parameterised lower bounds on the preprocessing and access time needed for solving DA tasks on signed queries. Having a better understanding of what happens on the fractional relaxation of  $\beta$ -hyperorder width would be a first step toward proving such lower bounds. Also, we believe our analysis of the complexity of DPLL is not optimal and that with the right data structures, we should be able to prove an upper bound of the order  $|\mathbf{D}|^{\text{fhow}(Q)}$  instead of the  $|\mathbf{D}|^{\text{fhow}(Q)+1}$  for positive queries, hence matching the existing upper bounds exactly. We leave a more involved analysis of this algorithm for future work. Finally, we believe that the circuit representation we are using is promising for answering different kind of aggregation tasks and hence generalising existing results to the case of signed conjunctive queries. For example, FAQ and AJAR queries [24, 21] could be answered using this data structure by annotating the circuit with semi-ring elements and

<sup>2</sup> The notion of  $S$ -connexity already exists for tree decompositions. We use the same name here as the existence of an  $S$ -connex tree decomposition of (fractional) hypertree width  $k$  is equivalent to the existence of an  $S$ -connex elimination order of (fractional) hyperorder width  $k$ .

projecting them out as in Theorem 19. Similarly, we believe that the framework of [15] for solving DA tasks on conjunctive queries with aggregation operators may be generalised in a similar way to the class of ordered  $\{\times, \text{dec}\}$ -circuits.

---

## References

- 1 Guillaume Bagan. *Algorithmes et complexité des problèmes d'énumération pour l'évaluation de requêtes logiques. (Algorithms and complexity of enumeration problems for the evaluation of logical queries)*. PhD thesis, University of Caen Normandy, France, 2009. URL: <https://tel.archives-ouvertes.fr/tel-00424232>.
- 2 Guillaume Bagan, Arnaud Durand, Etienne Grandjean, and Frédéric Olive. Computing the jth solution of a first-order query. *RAIRO-Theoretical Informatics and Applications*, 42(1):147–164, 2008. doi:10.1051/ita:2007046.
- 3 Nurzhan Bakibayev, Tomáš Kočiský, Dan Olteanu, and Jakub Závodný. Aggregation and ordering in factorised databases. *Proceedings of the VLDB Endowment*, 6(14):1990–2001, 2013. doi:10.14778/2556549.2556579.
- 4 Johann Brault-Baron. *De la pertinence de l'énumération: complexité en logiques propositionnelle et du premier ordre*. PhD thesis, Université de Caen, 2013. URL: <https://tel.archives-ouvertes.fr/tel-01081392>.
- 5 Johann Brault-Baron. Hypergraph acyclicity revisited. *ACM Computing Surveys (CSUR)*, 49(3):1–26, 2016. doi:10.1145/2983573.
- 6 Johann Brault-Baron, Florent Capelli, and Stefan Mengel. Understanding model counting for beta-acyclic CNF-formulas. In *32nd International Symposium on Theoretical Aspects of Computer Science*, volume 30 of *LIPICs*, pages 143–156. Schloss Dagstuhl, 2015. doi:10.4230/LIPICs.STACS.2015.143.
- 7 Karl Bringmann, Nofar Carmeli, and Stefan Mengel. Tight fine-grained bounds for direct access on join queries. In *Proceedings of the 41st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 427–436, 2022. doi:10.1145/3517804.3526234.
- 8 Karl Bringmann, Nofar Carmeli, and Stefan Mengel. Tight fine-grained bounds for direct access on join queries. *arXiv preprint arXiv:2201.02401*, 2022. doi:10.48550/arXiv.2201.02401.
- 9 Florent Capelli. Understanding the complexity of #SAT using knowledge compilation. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–10. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005121.
- 10 Florent Capelli and Oliver Irwin. Direct access for conjunctive queries with negation. *CoRR*, abs/2310.15800, 2023. doi:10.48550/arXiv.2310.15800.
- 11 Clément Carbonnel, Miguel Romero, and Stanislav Živný. Point-width and max-csps. *ACM Transactions on Algorithms (TALG)*, 16(4):1–28, 2020. doi:10.1145/3409447.
- 12 Nofar Carmeli, Nikolaos Tziavelis, Wolfgang Gatterbauer, Benny Kimelfeld, and Mirek Riedewald. Tractable orders for direct access to ranked answers of conjunctive queries. *ACM Transactions on Database Systems*, jan 2023. doi:10.1145/3578517.
- 13 Nofar Carmeli, Shai Zeevi, Christoph Berkholz, Benny Kimelfeld, and Nicole Schweikardt. Answering (unions of) conjunctive queries using random access and random-order enumeration. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 393–409, 2020. doi:10.1145/3375395.3387662.
- 14 Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing*, STOC '77, pages 77–90, New York, NY, USA, 1977. ACM. doi:10.1145/800105.803397.
- 15 Idan Eldar, Nofar Carmeli, and Benny Kimelfeld. Direct access for answers to conjunctive queries with aggregation. *arXiv preprint*, 2023. doi:10.48550/arXiv.2303.05327.

- 16 Johannes K Fichte, Markus Hecher, Neha Lodha, and Stefan Szeider. An smt approach to fractional hypertree width. In *Principles and Practice of Constraint Programming: 24th International Conference, CP 2018, Lille, France, August 27-31, 2018, Proceedings 24*, pages 109–127. Springer, 2018. doi:10.1007/978-3-319-98334-9\_8.
- 17 Robert Ganian, André Schidler, Manuel Sorge, and Stefan Szeider. Threshold treewidth and hypertree width. *Journal of Artificial Intelligence Research*, 74:1687–1713, 2022. doi:10.1613/jair.1.13661.
- 18 Georg Gottlob and Reinhard Pichler. Hypergraphs in model checking: Acyclicity and hypertree-width versus clique-width. *SIAM Journal on Computing*, 33(2):351–378, 2004. doi:10.1137/S0097539701396807.
- 19 Martin Grohe and Dániel Marx. Constraint solving via fractional edge covers. *ACM Transactions on Algorithms (TALG)*, 11(1):4, 2014. doi:10.1145/2636918.
- 20 Jinbo Huang and Adnan Darwiche. DPLL with a Trace: From SAT to Knowledge Compilation. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pages 156–162, 2005. URL: <http://ijcai.org/Proceedings/05/Papers/0876.pdf>.
- 21 Manas R Joglekar, Rohan Puttagunta, and Christopher Ré. Ajar: Aggregations and joins over annotated relations. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 91–106, 2016. doi:10.1145/2902251.2902293.
- 22 Jens Keppeler. *Answering Conjunctive Queries and FO+MOD Queries under Updates*. PhD thesis, Humboldt-Universität zu Berlin, Mathematisch-Naturwissenschaftliche Fakultät, 2020. URL: <http://edoc.hu-berlin.de/18452/22264>.
- 23 Mahmoud Abo Khamis, Hung Q Ngo, and Atri Rudra. Faq: questions asked frequently. *arXiv preprint*, 2015. doi:10.48550/arXiv.1504.04044.
- 24 Mahmoud Abo Khamis, Hung Q Ngo, and Atri Rudra. Faq: questions asked frequently. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 13–28, 2016. doi:10.1145/2902251.2902280.
- 25 Matthias Lanzinger. Tractability beyond  $\beta$ -acyclicity for conjunctive queries with negation and sat. *Theoretical Computer Science*, 942:276–296, 2023. doi:10.1016/j.tcs.2022.12.002.
- 26 Dan Olteanu. Factorized databases: A knowledge compilation perspective. In *AAAI Workshop: Beyond NP*, 2016. URL: <http://www.aaai.org/ocs/index.php/WS/AAAIW16/paper/view/12638>.
- 27 Dan Olteanu and Jakub Závodný. Factorised representations of query results: size bounds and readability. In *Proceedings of the 15th International Conference on Database Theory*, pages 285–298. ACM, 2012. doi:10.1145/2274576.2274607.
- 28 Dan Olteanu and Jakub Závodný. Size bounds for factorised representations of query results. *ACM Transactions on Database Systems (TODS)*, 40(1):1–44, 2015. doi:10.1145/2656335.
- 29 S. Ordyniak, D. Paulusma, and S. Szeider. Satisfiability of acyclic and almost acyclic CNF formulas. *Theoretical Computer Science*, 481:85–99, 2013. doi:10.1016/j.tcs.2012.12.039.
- 30 Reinhard Pichler and Sebastian Skritek. Tractable counting of the answers to conjunctive queries. *Journal of Computer and System Sciences*, 79:984–1001, sep 2013. doi:10.1016/j.jcss.2013.01.012.
- 31 Tian Sang, Fahiem Bacchus, Paul Beame, Henry A Kautz, and Toniann Pitassi. Combining component caching and clause learning for effective model counting. *Theory and Applications of Satisfiability Testing*, 2004. URL: <http://www.satisfiability.org/SAT04/programme/21.pdf>.
- 32 Maximilian Schleich, Dan Olteanu, and Radu Ciucanu. Learning linear regression models over factorized joins. In *Proceedings of the 2016 International Conference on Management of Data*, pages 3–18. ACM, 2016. doi:10.1145/2882903.2882939.
- 33 Mihalis Yannakakis. Algorithms for acyclic database schemes. In *Proceedings of the Seventh International Conference on Very Large Data Bases – Volume 7, VLDB ’81*, pages 82–94. VLDB Endowment, 1981.

# The Importance of Parameters in Database Queries

Martin Grohe  


RWTH Aachen University, Germany

Benny Kimelfeld  

Technion – Israel Institute of Technology, Haifa, Israel

Peter Lindner  

École Polytechnique Fédérale de Lausanne, Switzerland

Christoph Standke  

RWTH Aachen University, Germany

---

## Abstract

We propose and study a framework for quantifying the importance of the choices of parameter values to the result of a query over a database. These parameters occur as constants in logical queries, such as conjunctive queries. In our framework, the importance of a parameter is its SHAP score. This score is a popular instantiation of the game-theoretic Shapley value to measuring the importance of feature values in machine learning models. We make the case for the rationale of using this score by explaining the intuition behind SHAP, and by showing that we arrive at this score in two different, apparently opposing, approaches to quantifying the contribution of a parameter.

The application of the SHAP score requires two components in addition to the query and the database: (a) a probability distribution over the combinations of parameter values, and (b) a utility function that measures the similarity between the result for the original parameters and the result for hypothetical parameters. The main question addressed in the paper is the complexity of calculating the SHAP score for different distributions and similarity measures. We first address the case of probabilistically independent parameters. The problem is hard if we consider a fragment of queries that is hard to evaluate (as one would expect), and even for the fragment of acyclic conjunctive queries. In some cases, though, one can efficiently list all relevant parameter combinations, and then the SHAP score can be computed in polynomial time under reasonable general conditions. Also tractable is the case of full acyclic conjunctive queries for certain (natural) similarity functions. We extend our results to conjunctive queries with inequalities between variables and parameters. Finally, we discuss a simple approximation technique for the case of correlated parameters.

**2012 ACM Subject Classification** Theory of computation → Database query languages (principles)

**Keywords and phrases** SHAP score, query parameters, Shapley value

**Digital Object Identifier** 10.4230/LIPIcs.ICDT.2024.14

**Related Version** *Full Version*: <http://arxiv.org/abs/2401.04606> [11]

**Funding** The work of Martin Grohe, Benny Kimelfeld, and Christoph Standke was supported by the German Research Foundation (DFG) grants GR 1492/16-1 and KI 2348/1-1 (DIP Program). The work of Martin Grohe and Christoph Standke was supported by the German Research Foundation (DFG) grant GRK 2236/2 (UnRAVeL).

## 1 Introduction

The parameters of a database query may affect the result in a way that misrepresents the importance of the parameters, or the arbitrariness in their chosen values. For example, when searching for products in commercial applications (for clothing, travel, real estate, etc.), we may fill out a complex form of parameters that produce too few answers or overly expensive ones; what is the responsibility of our input values to this deficient outcome? We may phrase a database query to select candidates for awards for job interviews; to what extent is the choice of parameters affecting people’s fate?



© Martin Grohe, Benny Kimelfeld, Peter Lindner, and Christoph Standke;  
licensed under Creative Commons License CC-BY 4.0

27th International Conference on Database Theory (ICDT 2024).

Editors: Graham Cormode and Michael Shekelyan; Article No. 14; pp. 14:1–14:17

Leibniz International Proceedings in Informatics



Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Considerable effort has been invested in exploring the impact of parameters on query outcomes. In the *empty-answer problem*, the goal is typically to explore a space of small changes to the query that would yield a nonempty result [14, 22]. In that vein, reasoning about small parameter changes, or *perturbations*, has been applied to providing explanations to *non-answers*, that is, tuples that are missing from the result [5, 31]. From a different angle, the analysis of sensitivity to parameters has been applied to *fact checking*, and particularly, the detection of statements that are *cherry picked* in the sense that they lead to conclusions that overly rely on allegedly arbitrary parameter values. This may come in over-restriction to a database fragment that serves the intended claim [35], or over-generalization that masks the situation in substantial subgroups that oppose the claim [16].

In this work, we aim to establish a principled quantitative measure for the importance of individual parameter values to the result  $Q(D)$  of a query  $Q$  over a database  $D$ . To this end, we begin with the basic idea of observing how the result changes when we randomly change the parameter of interest. Alternatively, we can observe the change in the result when the parameter keeps its value while all others randomly change. Yet, these definitions ignore dependencies among parameters; changing a parameter may have no impact in the presence of other parameter values (e.g., the number of connecting flights does not matter if we restrict the travel duration), or it may lead to an overestimation of the value's importance (e.g., changing the number of semesters empties the result since we restrict the admission year). This can be viewed as a special case of a challenge that has been studied for decades in game theory: *How to attribute individual contributions to the gains of a team?* Specifically, we can view the parameter values as players of a cooperative game where each coalition (set of parameter values in our case) has a utility, and we wish to quantify the contribution of each parameter value to the overall utility. We then adopt a conventional formula for contribution attribution, namely the Shapley value [29], as done in many domains, including economics, law, bioinformatics, crime analysis, network analysis, machine learning, and more (see, e.g., the *Handbook of the Shapley value* [1]). The Shapley value is theoretically justified in the sense that it is unique under several elementary axioms of rationality for profit sharing [29]. In the context of databases, this value has been studied recently for measuring the contribution of individual tuples to query answers [17, 8, 2] and to database inconsistency [18], as well as the contribution of constraints to the decisions of cleaning systems [7].

Our challenge then boils down to one central question: What game are we playing? In other words, what is the utility of a set  $J$  of parameter values? Following up on the two basic ideas discussed above, we can think of two analogous ways. In the first way, we measure the change in the query result when we randomly change the values of the parameters in  $J$ ; the parameter values in  $J$  are deemed important if we observe a large change. This change is random, so we take the *expected* change. (We later discuss the way that we measure the change in the result.) In the second way, we again measure the change in the result, but now we do so when we fix the values of the parameters in  $J$  and randomly change the rest; now, however, the values in  $J$  are deemed important if we observe a *small* change, indicating that the other parameters have little impact once we use the values of  $J$ . This second way is known as the SHAP score [20, 19] in machine learning, and it is one of the prominent score-attribution methods for features (in addition to alternatives such as LIME [25] and Anchor [26]).<sup>1</sup> This score quantifies the impact of each feature value on the decision for a specific given instance. Interestingly, we show that the first way described above also coincides

---

<sup>1</sup> For background on attribution scores see textbooks on explanations in machine learning, e.g., Molnar [21].



with the SHAP score, so the two ways actually define the *same measure* (Theorem 3.4). We prove it in a general setting and, hence, this equivalence is of independent interest as it shows an alternative, apparently different way of arriving at SHAP in machine learning.

To materialize the framework in the context of a query  $Q$  and a database  $D$ , one needs to provide some necessary mechanisms for reasoning about  $Q(D)$  and  $Q'(D)$ , where  $Q'$  is the same as  $Q$  up to the parameters: it uses the same values as  $Q$  for the parameters of  $J$ , but the remaining parameter values are selected randomly. Specifically, to this aim, we need two mechanisms:

1. A *probability distribution*  $\Gamma$  of possible parameterizations of the query;
2. A *similarity function*  $\mathfrak{s}$  between relations to quantify how close  $Q'(J)$  is to  $Q(J)$ .

The distribution  $\Gamma$  may include any feasible combination of parameter values, and they can be either probabilistically independent or correlated. For  $\mathfrak{s}$ , one can use any similarity between sets (see, e.g., surveys on similarity measures such as [15, 28]) or measures that account for the distance between attributes values (e.g., as done in the context of database repairs [4]). We give examples in Section 3.

A central challenge in the framework is the computational complexity, since the direct definition of the SHAP score (like the general Shapley value) involves summation over an exponential space of coalitions. Indeed, the calculation can be a hard computational problem,  $\#P$ -hard to be precise, even for simple adaptations of the SHAP score [3] and the Shapley value [10, 6, 17]. Hence, instantiations of our framework require specialized complexity analyses and nontrivial algorithms that bypass the exponential time of the naïve computation.

We begin the complexity analysis of the framework by establishing some general insights for finite fully factorized distributions, i.e., where the parameters are probabilistically independent and each is given as an explicit collection of value-probability pairs. First, the SHAP score can be computed in polynomial time if we can evaluate the query, compute the similarity measure, and enumerate all parameter combinations in polynomial time. We prove this using a recent general result by Van den Broeck et al. [32] (from which we borrow some notation) showing that, under tractability assumptions, the SHAP score is reducible to the computation of the expected value under random parameter values. Second, under reasonable assumptions, the computation of the SHAP score is at least as hard as testing for the emptiness of the query, for *every* nontrivial similarity function; this is expected, as the definition of the SHAP scores requires, conceptually, many applications of the query.

Next, we focus on the class of conjunctive queries, where the parameters are constants in query atoms. Put differently, we consider Select-Project-Join queries where each selection predicate has the form  $x = p$  where  $x$  is an attribute and  $p$  is a parameter. It follows from the above general results that this case is tractable under data complexity. Hence, we focus on combined complexity. As the emptiness problem is intractable, we consider the tractable fragment of acyclic queries, and show that SHAP scores can be  $\#P$ -hard even there.

We then focus on the class of *full* acyclic queries and establish that the SHAP score can be computed in polynomial time for three natural, set-based similarity functions between  $Q(D)$  and  $Q'(D)$ . Interestingly, this gives us nontrivial cases where the SHAP score can be computed in polynomial time even if  $Q(D)$  and  $Q'(D)$  can be exponential in the size of the input, and hence, it is intractable to materialize them.

We then extend our results to conjunctive queries with inequalities, that is, built-in atoms of the form  $x \leq p$  where  $x$  is a variable and  $p$  is a parameter. We show that this addition can make the SHAP score intractable to compute, even if there is a single relational atom in addition to the inequality atoms. Nevertheless, we identify cases in which tractability properties are retained when adding inequalities to classes of parameterized queries (e.g., full acyclic conjunctive queries), relying on structural assumptions on the use of inequalities.



Given that the computation of the exact SHAP score is often intractable, we also study the complexity of approximate evaluation. We show that using sampling, we can obtain an efficient approximation scheme (FPRAS) with additive guarantees. Moreover, the tractability of approximation generalizes to allow for parameters that are correlated through Bayesian networks (and actually any distribution) that provide(s) polynomial-time sampling while conditioning on assignments to arbitrary subsets of the random variables.

Details omitted due to space limitations, including the full proofs of the proof sketches, can be found in the full version of the paper [11].

## 2 Preliminaries

We write  $2^S$  for the power set of  $S$ . Vectors, tuples and sequences are denoted by boldface letters. If  $\mathbf{x} = (x_i)_{i \in I}$  and  $J \subseteq I$ , then  $\mathbf{x}_J = (x_j)_{j \in J}$ . Moreover,  $|\mathbf{x}| = |I|$  is the number of entries of  $\mathbf{x}$ . We let  $[n] = \{1, \dots, n\}$ . Truth values (true/false) are denoted by **tt** and **ff**.

A (discrete) *probability distribution* is a function  $\Gamma: S \rightarrow [0, 1]$ , where  $S$  is a non-empty countable set, and  $\sum_{s \in S} \Gamma(s) = 1$ . The *support* of  $\Gamma$  is  $\text{supp}(\Gamma) = \{s \in S: \Gamma(s) > 0\}$ . We use  $\Pr$  for generic probability distributions and, in particular,  $\Pr_{X \sim \Gamma}$  to refer to probabilities for a random variable or random vector  $X$  being drawn from the distribution  $\Gamma$ . Likewise,  $\mathbb{E}_{X \sim \Gamma}$  refers to the expectation operator with respect to the distribution  $\Gamma$  of  $X$ .

Whenever  $x$  is an input to a computational problem, we write  $\|x\|$  to refer to the encoding length of  $x$  as it is represented in the input.

### 2.1 Schemas and Databases

A *relation schema* is a sequence  $\mathbf{A} = (A_1, \dots, A_k)$  of distinct *attributes*  $A_i$ , each with an associated *domain*  $\text{dom}(A_i)$  of values. We call  $k$  the *arity* of  $\mathbf{A}$ . If  $\mathbf{A} = (A_1, \dots, A_k)$  is a relation schema and  $S = \{i_1, \dots, i_\ell\} \subseteq [k]$  with  $i_1 < i_2 < \dots < i_\ell$ , then  $\mathbf{A}_S$  denotes the relation schema  $(A_{i_1}, \dots, A_{i_\ell})$ .

A *tuple* over  $\mathbf{A}$  is an element  $\mathbf{a} = (a_1, \dots, a_k) \in \text{dom}(A_1) \times \dots \times \text{dom}(A_k)$ . The set of tuples over  $\mathbf{A}$  is denoted by  $\text{Tup}[\mathbf{A}]$ . A *relation* over  $\mathbf{A}$  is a finite set of tuples over  $\mathbf{A}$ . We denote the space of relations over  $\mathbf{A}$  by  $\text{Rel}[\mathbf{A}]$ . We typically denote relations by  $T$  or  $T_1, T_2$ , and so on. *By default, we assume that all attribute domains are countably infinite.*

A *database schema* is a finite set of *relation symbols*, where every relation symbol is associated with a relation schema. For example, if  $R$  is a relation symbol with associated relation schema  $\mathbf{A} = (A_1, \dots, A_k)$ , we may refer to  $R$  by  $R(\mathbf{A})$  or  $R(A_1, \dots, A_k)$ , and call  $k$  the *arity* of  $R$ . A *fact*  $f$  over a database schema  $\mathbf{S}$  is an expression of the shape  $R(\mathbf{a})$  with  $R = R(\mathbf{A})$ , where  $\mathbf{a}$  is a tuple over  $\mathbf{A}$ . A *database* over schema  $\mathbf{S}$  is a finite set of facts over  $\mathbf{S}$ . We denote the space of databases over some schema  $\mathbf{S}$  by  $\text{DB}[\mathbf{S}]$ .

### 2.2 Queries and Parameters

A *relational query* (or just *query*) is an isomorphism invariant function that maps databases to relations. More precisely, a query  $q$  has a database schema  $\text{dom}(q)$  for its valid input databases (called the *domain schema*), and a relation schema  $\text{range}(q)$  for its output relation (called the *range schema*). Then  $q$  is a function that maps databases over  $\text{dom}(q)$  to relations over  $\text{range}(q)$ . We write  $q = q(\mathbf{R})$  to denote that  $\text{range}(q) = \mathbf{R}$ .

We focus on queries that are expressed in variants of first-order logic without equality. In this case,  $\text{range}(q)$  is the domain of the free variables of  $q$ . *Whenever we discuss queries in this paper, it is assumed that they are first-order queries, unless explicitly stated*

otherwise. To emphasize that  $q$  has free variables  $\mathbf{x} = (x_1, \dots, x_n)$ , we write  $q = q(\mathbf{x})$ . If  $\mathbf{a} = (a_1, \dots, a_n) \in \text{Tuple}[\mathbf{R}]$ , then  $q(\mathbf{a})$  denotes the Boolean query obtained from  $q$  by substituting every occurrence of  $x_i$  with  $a_i$ ,  $i \in [n]$ . If  $D$  is a database over  $\text{dom}(q)$ , then  $q(D) := \{\mathbf{a} \in \text{Tuple}[\mathbf{R}]: D \models q(\mathbf{a})\}$ .

A *parameterized query*  $Q$  is a relational query in which some of the free variables are distinguished *parameters*. We write  $Q(\mathbf{x}; \mathbf{y})$  to denote that  $\mathbf{x}$  are the non-parameter (free) variables of  $Q$ , and that  $\mathbf{y}$  are the parameters of  $Q$ . A parameterized query  $Q$  has a *parameter schema*  $\text{param}(Q)$ , such that  $\text{Tuple}[\text{param}(Q)]$  is the set of valid tuples of parameter values for  $Q$ . If  $Q = Q(\mathbf{x}; \mathbf{y})$  is a parameterized query, and  $\mathbf{p} \in \text{Tuple}[\text{param}(Q)]$ , then  $Q_{\mathbf{p}}(\mathbf{x}) := Q(\mathbf{x}; \mathbf{p})$  is the relational query obtained from  $Q$  by substituting the parameters  $\mathbf{y}$  with the constants  $\mathbf{p}$ . The *domain* and *range schemas* of  $Q$  coincide with  $\text{dom}(Q_{\mathbf{p}})$  and  $\text{range}(Q_{\mathbf{p}})$ , respectively, which is independent of the choice of  $\mathbf{p} \in \text{Tuple}[\text{param}(Q)]$ . We write  $Q = Q(\mathbf{R}; \mathbf{P})$  to denote that  $\text{range}(Q) = \mathbf{R}$  and  $\text{param}(Q) = \mathbf{P}$ . If  $Q = Q(\mathbf{R}; \mathbf{P})$ , then for every database  $D$  over  $\text{dom}(Q)$  and every  $\mathbf{p} \in \text{Tuple}[\mathbf{P}]$  we have

$$Q_{\mathbf{p}}(D) := \{\mathbf{a} \in \text{Tuple}[\mathbf{R}]: D \models Q_{\mathbf{p}}(\mathbf{a})\} \in \text{Rel}[\mathbf{R}].$$

► **Example 2.1.** For illustration, we consider a parameterized query over flights data, assuming access to a database with relations `Flight`(`Id`, `Date`, `AirlineName`, `From`, `To`, `Departure`, `Arrival`) and `Airline`(`Name`, `CountryOfOrigin`). Then

$$Q(f, t_{\text{dep}}, t_{\text{arr}}; d, c) = \exists a: \text{Flights}(f, d, a, \text{CDG}, \text{JFK}, t_{\text{dep}}, t_{\text{arr}}) \wedge \text{Airline}(a, c)$$

is a parameterized query. For parameters  $(d, c)$  it asks for departure and arrival times of flights from Paris to New York City on date  $d$  operated by an airline from country  $c$ . ◻

Parameterized queries inherit structural properties from the underlying relational query. For example, if  $Q(\mathbf{x}; \mathbf{y})$  is a parameterized query with free variables  $\mathbf{x}$ , then  $Q$  is called an (acyclic) conjunctive query, if the relational query  $Q(\mathbf{x}, \mathbf{y})$  with variables  $\mathbf{x}, \mathbf{y}$  is an (acyclic) conjunctive query. We emphasize that for the notion of acyclicity, the parameters act as additional free variables. However, a parameterized query  $Q$  is called *Boolean*, if all (non-parameter) variables in  $Q$  are quantified, i.e., if  $Q_{\mathbf{p}}$  is Boolean for all  $\mathbf{p}$ .

### 2.3 The Shapley Value

A *cooperative game* is a pair  $(I, \nu)$  where  $I$  is a non-empty set of players, and  $\nu: 2^I \rightarrow \mathbb{R}$  is a *utility function* that assigns a “joint wealth”  $\nu(J)$  to every *coalition*  $\nu(J) \subseteq I$ . The *Shapley value* of player  $i \in I$  in  $(I, \nu)$  is defined as

$$\text{Shapley}(I, \nu, i) := \frac{1}{|I|!} \sum_{\sigma \in S_I} (\nu(\sigma_i \cup \{i\}) - \nu(\sigma_i)) = \mathbb{E}_{\sigma \sim S_I} [\nu(\sigma_i \cup \{i\}) - \nu(\sigma_i)], \quad (1)$$

where  $S_I$  is the set of permutations of  $I$ , and  $\sigma_i$  is the set of players appearing before  $i$  in the permutation  $\sigma \in S_I$  (and  $\sigma \sim S_I$  refers to drawing from the uniform distribution on  $S_I$ ). Intuitively,  $\text{Shapley}(I, \nu, i)$  measures the importance (or, contribution) of  $i$  among coalitions. An equivalent, alternative expression is

$$\text{Shapley}(I, \nu, i) = \sum_{J \subseteq I \setminus \{i\}} \Pi_i(J) \cdot (\nu(J \cup \{i\}) - \nu(J)) = \mathbb{E}_{J \sim \Pi_i} [\nu(J \cup \{i\}) - \nu(J)], \quad (2)$$

where  $\Pi_i$  is the probability distribution on  $I \setminus \{i\}$  with

$$\Pi_i(J) = \frac{|J|! \cdot (|I| - |J| - 1)!}{|I|!} = \frac{1}{|I| \cdot \binom{|I|-1}{|J|}}. \quad (3)$$

(For further details, see textbooks on the Shapley value, e.g., [1].) In the next section, we will discuss the intuition behind using the Shapley value in the context of our paper.

### 3 The SHAP Score of Query Parameters

In this section, we shall give our exact definition of the SHAP score of a query parameter, and we will argue that it measures the contribution of the parameter to the outcome of the query in a natural way. We start with a database  $D$  over  $\text{dom}(Q)$ ; a parameterized query  $Q(\mathbf{R}; \mathbf{P})$ ; and a *reference parameter*  $\mathbf{p}^* = (p_1^*, \dots, p_\ell^*)$  over  $\mathbf{P}$ . We want to quantify the contribution of each parameter  $p_i^*$  to the query answer  $Q_{\mathbf{p}^*}(D)$ .

#### 3.1 Intuition

A basic idea is to see what happens to the answer if we either modify  $p_i^*$  and keep all other parameters fixed or, conversely, keep  $p_i^*$  fixed but modify the other parameters. The following example shows that neither of these two approaches is sufficient.

► **Example 3.1.** We let  $Q(x; y_1, y_2, y_3) := R(y_1, y_2, y_3, x)$  for a relation  $R(B_1, B_2, B_3, A)$ . Let  $N := [n] = \{1, \dots, n\}$ , and  $D$  be the database where the tuple set of  $R$  is

$$((\{1\} \times N) \cup (N \times \{1\})) \times N \times N.$$

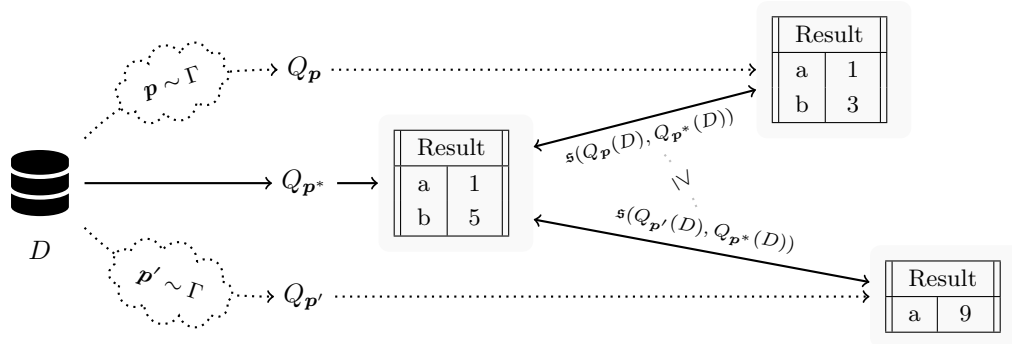
Let  $\mathbf{p}^* = (p_1^*, p_2^*, p_3^*) = (1, 1, 1)$ . Then  $Q_{\mathbf{p}^*}(D) = N$ .

Let us try to understand the contribution of each parameter  $p_i^*$ . Intuitively, the parameter  $p_3^*$  is completely irrelevant. Whatever value it takes in any configuration of the other two parameters, the query answer remains the same. Parameters  $p_1^*$  and  $p_2^*$  are important, though. If we change both of their values to values  $p_1, p_2 \in N \setminus \{1\}$ , the query answer becomes empty. However, for this change to take effect, *we need to change both values at the same time*. If we change only  $p_1$  keeping  $p_2^*$  and  $p_3^*$  fixed, the query answer does not change. If we keep  $p_1^*$  fixed and modify  $p_2$  and  $p_3$ , then again the query answer does not change. The same holds for the second parameter. ┘

What this example shows is that even if we just want to understand the contribution of the individual parameters, we need to look at the contributions of *sets* of parameters. This will immediately put us in the realm of *cooperative game theory* with the parameters acting as players in a coalitional game.

Assume for a moment that we have a function  $\nu : 2^{[\ell]} \rightarrow \mathbb{R}$  such that, for  $J \subseteq [\ell]$ , the function value  $\nu(J)$  quantifies the combined contribution of the parameters with indices in  $J$ . We will discuss later how to obtain such a function  $\nu$ .

Given  $\nu$ , we now quantify the contribution of every *individual* parameter  $j \in [\ell]$ . This is exactly what the Shapley value  $\text{Shapley}([\ell], \nu, j)$  does. And not only that: as Shapley [29] proved, subject to a few natural assumptions (“axioms”) and the technical requirement that  $\nu$  is zero on the empty set, it is the *only* way of doing this. Let us briefly review these axioms (for details and a proof of Shapley’s theorem, we refer to the literature, e.g., [29, 27, 1]). The first axiom, *symmetry*, says that if any two parameters  $i$  and  $j$  (or rather parameter indices) have the same contribution to any coalition  $J \subseteq [\ell] \setminus \{i, j\}$ , then  $\text{Shapley}([\ell], \nu, i) = \text{Shapley}([\ell], \nu, j)$ . The second axiom, *efficiency*, says that the sum of  $\text{Shapley}([\ell], \nu, j)$  for all parameters  $j \in [\ell]$  equals the contribution of the “grand coalition”  $[\ell]$ . Thus, the Shapley values “distribute”  $\nu([\ell])$  among the individual parameters. The third axiom, *null player*, says that if a parameter makes no contribution to any coalition, then  $\text{Shapley}([\ell], \nu, j) = 0$ . The fourth axiom, *additivity*, says that if we have two different functions  $\nu_1, \nu_2 : 2^{[\ell]} \rightarrow \mathbb{R}$  measuring the value of a coalition of parameters, then  $\text{Shapley}([\ell], \nu_1 + \nu_2, j) = \text{Shapley}([\ell], \nu_1, j) + \text{Shapley}([\ell], \nu_2, j)$ . All of the axioms strike us as completely natural also for evaluating the contribution of parameters to a query answer.



**Figure 1** On a given database  $D$ , we evaluate a parameterized query  $Q$  with respect to a reference parameter  $p^*$ . The result is compared (using a similarity function  $s$ ) against different parameterizations of  $Q$ , according to a probabilistic model  $\Gamma$ .

Concluding the discussion so far, to understand the contribution of individual parameters, we need to look at sets of parameters, and the method of choice to retrieve this contribution from a valuation for sets is the Shapley value.

### 3.2 The Utility Function

What remains to be done is constructing a natural valuation for sets of parameters, that is, a function  $\nu : 2^{[\ell]} \rightarrow \mathbb{R}$  such that, for  $J \subseteq [\ell]$  the value  $\nu(J)$  quantifies the combined contribution of the parameters  $p_j^*$ , for  $j \in J$ , to the query answer. The first thing we need is a way of measuring the similarity between query answers, that is, a function  $s$  that takes two relations of the schema of our query answer and assigns a similarity value to them. Formally, a *similarity function* is just an arbitrary function  $s : \text{DB}[\mathbf{R}] \times \text{DB}[\mathbf{R}] \rightarrow \mathbb{R}$ . We make no restrictions on this function, similarity can have different meanings depending on the application. Let us just note that the intended use of this function is always to quantify similarity: the higher  $s(Q, Q^*)$ , the more similar  $Q$  and  $Q^*$  are.

► **Example 3.2.**

- (a) Prime examples of similarity functions are set-similarity measures such as the *Jaccard index*, the *Sørensen index*, and *Tverski's index*, which attempt to capture the degree of similarity between two sets [23]. If  $Q$  and  $Q^*$  are two relations of the same relation schema  $\mathbf{R}$ , then, e.g., their Jaccard index is given by

$$\text{Jaccard}(Q, Q^*) := \frac{|Q \cap Q^*|}{|Q \cup Q^*|} = 1 - \frac{|Q \Delta Q^*|}{|Q \cup Q^*|},$$

and  $\text{Jaccard}(\emptyset, \emptyset) = 0$  by convention.

- (b) In the same spirit, but somewhat simpler, are similarities based on cardinalities and basic set operations, for example:

$$\begin{aligned} \text{Int}(Q, Q^*) &:= |Q \cap Q^*|, && \text{(size of the intersection)} \\ \text{NegSymDiff}(Q, Q^*) &:= -|Q \Delta Q^*|, && \text{(negative symmetric difference)} \\ \text{NegSymCDiff}(Q, Q^*) &:= -||Q| - |Q^*||. && \text{(negative symmetric cardinality difference)} \end{aligned}$$

Note that we do not require similarity to be non-negative, thus  $\text{NegSymDiff}$  and  $\text{NegSymCDiff}$  are well-defined. Intuitively, the negative sign makes sense because two relations get more similar as their symmetric difference gets smaller.

- (c) In our applications,  $Q$  and  $Q^*$  have different roles:  $Q^*$  is always a fixed “reference relation”, and  $Q$  is a modification of  $Q^*$ , obtained by changing some parameters in the query. In some situations, it can be useful to use similarity measures that are not symmetric. For example, we may only care about not losing any tuples from  $Q^*$ , whereas additional tuples may be irrelevant. This leads to an asymmetric similarity measure:

$$\text{NegDiff}(Q, Q^*) := -|Q^* \setminus Q|. \quad (\text{negative difference})$$

We can also use similarity functions that are specialized to the application at hand. For example, we can use a similarity function that accounts for differences of attribute values, such as arrival time minus departure time:

$$\text{MinDiff}_{A,B}(Q, Q^*) := -|\min(Q.A - Q.B) - \min(Q^*.A - Q^*.B)|.$$

where  $Q.A$  is the vector obtained from numerical  $A$  column of  $Q$  (hence,  $Q.A - Q.B$  is the vector of differences between the  $A$  and  $B$  attributes). Alternatively, we can use

$$\text{ExpMinDiff}_{A,B}(Q, Q^*) := \exp(\min(Q.A - Q.B) - \min(Q^*.A - Q^*.B)).$$

which uses the exponent function instead of the absolute value.  $\lrcorner$

Suppose now that we have a subset  $J \subseteq [\ell]$ . We need a value  $\nu(J)$  quantifying how important the subtuple  $\mathbf{p}_J^* = (p_j)_{j \in J}$  is to obtain a query answer similar to  $Q_{\mathbf{p}^*}(D)$ . The crucial idea of the SHAP score in a very similar setting in machine learning [20] was to see what happens if we fix the parameters  $\mathbf{p}_J^*$  and change the other parameters: the value of  $J$  is high if changing the other parameters has no big effect on the query answer. In other words, the value of  $J$  is large if for parameter tuples  $\mathbf{p}$  with  $\mathbf{p}_J = \mathbf{p}_J^*$  the relations  $Q_{\mathbf{p}}(D)$  and  $Q_{\mathbf{p}^*}(D)$  are similar, that is,  $\mathfrak{s}(\mathbf{p}, \mathbf{p}^*)$  is large.<sup>2</sup> But of course this depends on how exactly we change the other parameters. The idea is to change them randomly and take the expected value, as illustrated in Figure 1. So we assume that we have a probability distribution  $\Gamma$  on the space of all possible parameter values, and we define

$$\nu(J) := \mathbb{E}_{\mathbf{p} \sim \Gamma} [\mathfrak{s}(\mathbf{p}, \mathbf{p}^*) \mid \mathbf{p}_J = \mathbf{p}_J^*] = \mathbb{E}_{\mathbf{p} \sim \Gamma} [\mathfrak{s}(Q_{\mathbf{p}}(D), Q_{\mathbf{p}^*}(D)) \mid \mathbf{p}_J = \mathbf{p}_J^*].$$

► Remark 3.3. Officially, we should “normalize” to  $\nu(\emptyset) = 0$  by subtracting  $\mathbb{E}_{\mathbf{p}} [\mathfrak{s}(\mathbf{p}, \mathbf{p}^*)]$ . We chose not to do so, because it leads to the same Shapley values due to linearity.  $\lrcorner$

With the above, we can quantify the contribution of any individual parameter  $i$  as Shapley  $([\ell], \nu, i)$ . In using different probability distributions, we can incorporate various assumptions about the usage of parameters. For example,  $\Gamma$  may apply to only a subset of interesting parameters and be deterministic (fixed) on the others; it can also be concentrated on certain values (e.g., locations of a reasonable distance to the original value) or describe any other statistical model on the parameter space.

There is a different, in some way complementary approach to quantifying the contribution of a set  $J$  of parameters: instead of measuring what happens if we fix  $\mathbf{p}_J^*$  and randomly change the other parameters, we can also fix the other parameters and randomly change the parameters in  $J$ . Then the contribution of  $J$  is the higher, the more the query answer changes. Thus, now instead of our similarity measure  $\mathfrak{s}$  we need a *dissimilarity measure*  $\bar{\mathfrak{s}}$ . We simply let  $\bar{\mathfrak{s}}(\mathbf{p}, \mathbf{p}') := c - \mathfrak{s}(\mathbf{p}, \mathbf{p}')$ , where  $c \in \mathbb{R}$  is an arbitrary constant. (We will see that this

<sup>2</sup> We use  $\mathfrak{s}(\mathbf{p}, \mathbf{p}^*)$  as shorthand for  $\mathfrak{s}(Q_{\mathbf{p}}(D), Q_{\mathbf{p}^*}(D))$  in case  $Q$  and  $D$  are clear from the context.

constant is completely irrelevant, but it may have some intuitive meaning. For example, if our similarity measure is normalized to take values in the interval  $[0, 1]$ , as Jaccard similarity, then it would be natural to take  $c = 1$ .) Letting  $\bar{J} := [\ell] \setminus J$  for every  $J \subseteq [\ell]$ , our new value functions for sets of indices is

$$\bar{\nu}(J) := \mathbb{E}_{\mathbf{p} \sim \Gamma} [\bar{\mathfrak{s}}(\mathbf{p}, \mathbf{p}^*) \mid \mathbf{p}_{\bar{J}} = \mathbf{p}_{\bar{J}}^*].$$

We could have started our treatment with introducing  $\bar{\nu}$  instead of  $\nu$ ; indeed, the only reason that we did not is that the analogue of  $\nu$  is what is used in machine learning. Both  $\nu$  and  $\bar{\nu}$  strike us as completely natural value functions for sets of parameters, and we see no justification for preferring one over the other. Fortunately, we do not have to, because they both lead to exactly the same Shapley values for the individual parameters.

► **Theorem 3.4.** *For all  $i \in [\ell]$  we have  $\text{Shapley}([\ell], \nu, i) = \text{Shapley}([\ell], \bar{\nu}, i)$ .*

**Proof sketch.** Linearity of expectation entails  $\nu(J) = c - \bar{\nu}(\bar{J})$ . The main observation is then on the relationship between  $J$  and  $\bar{J} \cup \{i\}$ , for  $J \subseteq [\ell] \setminus \{i\}$ : Both sets have the same probability under  $\Pi_i$  and, moreover, there is a one-to-one correspondence between them. Changing the summation accordingly shows that the SHAP scores coincide. ◀

When  $Q, D, \mathbf{p}^*, \mathfrak{s}, \Gamma$  are clear from the context, we write  $\text{SHAP}(i) := \text{Shapley}([\ell], \nu, i)$ ; hence:

$$\begin{aligned} \text{SHAP}(i) = & \\ & \mathbb{E}_{J \sim \Pi_i} \left[ \mathbb{E}_{\mathbf{p} \sim \Gamma} [\mathfrak{s}(Q_{\mathbf{p}}(D), Q_{\mathbf{p}^*}(D)) \mid \mathbf{p}_{J \cup \{i\}} = \mathbf{p}_{J \cup \{i\}}^*] - \mathbb{E}_{\mathbf{p} \sim \Gamma} [\mathfrak{s}(Q_{\mathbf{p}}(D), Q_{\mathbf{p}^*}(D)) \mid \mathbf{p}_J = \mathbf{p}_J^*] \right]. \end{aligned}$$

We illustrate the framework using our running example from Section 2.

► **Example 3.5.** In Example 2.1, we considered a flights database  $D$  and a query  $Q$ , parameterized with a date  $d$  and an airline country  $c$ , asking for flights between Paris and New York City. We assess the importance of  $Q$ 's parameters with respect to the reference parameter  $d = 02/24/2024$  and  $c = \text{USA}$ .

In specifying the parameter distribution  $\Gamma$ , we can tune the exploration of the parameter space. For example, we may choose to only take local parameter perturbations into account. For this, we can specify  $\Gamma$  to be a product distribution whose support is restricted to dates in the vicinity of  $02/24/2024$ , and to the countries **USA** and **France**. If the similarity function `NegSymCDiff` is used, e.g., then it is measured how close  $Q_{\mathbf{p}}$  and  $Q_{\mathbf{p}^*}$  are in terms of the number of given flight options. If `NegDiff` is used, it is measured how many of  $Q_{\mathbf{p}^*}$ 's flight options are lost when the parameters are changed. With `MinDiffArrival, Departure`, the difference in duration of the shortest flight options is assessed. ◻

### 3.3 Formal Problem Statement

We now have all the ingredients to formulate the computation of SHAP scores as a formal computational problem. It can be phrased concisely as follows.

► **Problem 3.6.**  $\text{SHAP}(\mathcal{Q}, \text{PR}, \mathfrak{s})$  is the following computational problem:

On input  $(Q, \mathbf{p}^*, D, \Gamma)$ , compute  $\text{SHAP}(i)$  for all  $i \in [\ell]$ . ◻

Table 1 presents an overview of the problem parameters and inputs. For technical reasons (which we explain next), the problem is parameterized with a class of parameterized queries  $\mathcal{Q}$ , a class of parameter distributions  $\text{PR}$ , and a (class of) similarity function(s)  $\mathfrak{s}$ .

■ **Table 1** Problem parameters and inputs for SHAP score computation.

Problem parameters		Problem inputs	
$\mathcal{Q}$	class of parameterized queries	$Q(\mathbf{R}; \mathbf{P})$	parameterized query
$\text{PR}$	class of parameter distributions	$D \in \text{DB}[\text{dom}(Q)]$	database
$\mathfrak{s}$	similarity function	$\mathbf{p}^* = (p_1^*, \dots, p_\ell^*) \in \mathbf{P}$	reference parameter
		$\Gamma \in \text{PR}_{\mathbf{P}}, \Gamma(\mathbf{p}^*) > 0$	parameter distribution

**Parameter distributions.** For simplicity, we only consider parameter distributions with finite support and rational probabilities. A class of parameter distributions will always mean a class  $\text{PR} = \bigcup_{\mathbf{P}} \text{PR}_{\mathbf{P}}$  where  $\mathbf{P}$  are relation schemas, and  $\text{PR}_{\mathbf{P}}$  is a set of functions  $\Gamma : \text{Tup}[\mathbf{P}] \rightarrow [0, 1]$  such that  $\sum_{\mathbf{p}} \Gamma(\mathbf{p}) = 1$ .

► **Example 3.7.** One of the simplest classes of parameter distributions is the class  $\text{IND}$  of *fully factorized distributions*, where all parameters are stochastically independent. A probability distribution  $\Gamma$  over  $\text{Tup}[\mathbf{P}]$  is called *fully factorized* if

$$\Gamma(p_1, \dots, p_\ell) = \Gamma_1(p_1) \cdots \Gamma_\ell(p_\ell)$$

for all  $(p_1, \dots, p_\ell) \in \text{Tup}[\mathbf{P}]$  where  $\Gamma_i$  is the marginal distribution of  $p_i$  under  $\Gamma$ . To represent  $\Gamma$ , it suffices to provide the  $\ell$  lists of all pairs  $(p_i, \Gamma_i(p_i))$  where  $\Gamma_i(p_i) > 0$ . ◀

Technically,  $\text{PR}$  is a class of *representations* of probability distributions in the SHAP problem. For convenience, we do not distinguish between a distribution and (one of) its representation. The only assumption we *globally* make about classes of parameterized distributions is that the encoding length of parameters is polynomially bounded in  $\|\Gamma\|$ .

**Similarity functions.** It would feel natural to consider  $\mathfrak{s}$  as an input to the problem. Discussing computational complexity in terms of an encoding of such functions is, however, beyond the scope of this paper. We therefore assume, that the similarity function (or rather, a class  $(\mathfrak{s}_{\mathbf{R}})_{\mathbf{R}}$  of versions of the same abstract similarity function  $\mathfrak{s}$ , one per query range schema) is a parameter to the problem. We abuse notation, and call this class  $\mathfrak{s}$  too. Moreover, we write  $\mathfrak{s}(T_1, T_2)$  for  $\mathfrak{s}_{\mathbf{R}}(T_1, T_2)$  when  $T_1, T_2 \in \text{DB}[\mathbf{R}]$ , and  $\mathbf{R}$  is clear from the context.

**Parameterized queries.** The parameterized queries we consider are formulated in first-order logic, unless explicitly stated otherwise. The encoding size of  $Q_{\mathbf{p}}$  can be larger than that of the parameterized query  $Q$ . By our assumptions on  $\Gamma$ , this blow-up is at most polynomial in  $\|Q\|$  and  $\|\Gamma\|$  if  $\mathbf{p} \in \text{supp}(\Gamma)$ .

**Complexity.** We abuse notation and use  $\mathfrak{s} \circ \mathcal{Q}$  to denote the class of functions mapping  $(Q, \mathbf{p}, \mathbf{p}^*, D)$  to  $\mathfrak{s}(Q_{\mathbf{p}}(D), Q_{\mathbf{p}^*}(D))$  where  $Q = Q(\mathbf{R}; \mathbf{P}) \in \mathcal{Q}$ , where  $D \in \text{DB}[\text{dom}(Q)]$ , and where  $\mathbf{p}, \mathbf{p}^* \in \text{Tup}[\mathbf{P}]$ .

► **Definition 3.8.** Let  $\mathcal{Q}$  be a class of parameterized queries,  $\mathfrak{s}$  a similarity function. We call

- $\mathcal{Q}$  tractable, if  $(Q, \mathbf{p}, D) \mapsto Q_{\mathbf{p}}(D)$  can be computed in polynomial time.
- $\mathfrak{s}$  tractable, if  $(T_1, T_2) \mapsto \mathfrak{s}(T_1, T_2)$ , can be computed in polynomial time.
- $\mathfrak{s} \circ \mathcal{Q}$  tractable, if  $(Q, \mathbf{p}, \mathbf{p}^*, D) \mapsto \mathfrak{s}(Q_{\mathbf{p}}(D), Q_{\mathbf{p}^*}(D))$  can be computed in polynomial time.

Tractability of both  $\mathfrak{s}$  and  $\mathcal{Q}$  imply tractability of  $\mathfrak{s} \circ \mathcal{Q}$ , but the converse is not true in general (see [12]). For illustration of this phenomenon, consider the query answering problem for full ACQs: The query answer may be exponentially large, but the answer *count* can be computed efficiently [24, Theorem 1].



► **Definition 3.9.** Let  $\mathcal{Q}$  be a class of parameterized queries,  $\text{PR}$  a class of parameter distributions, and  $\mathfrak{s}$  a similarity function, and let  $\mathcal{C}$  be a complexity class.

- If the complexity of solving  $\text{SHAP}(\mathcal{Q}, \text{PR}, \mathfrak{s})$ , as a function of  $\|Q\| + \|\mathbf{p}^*\| + \|D\| + \|\Gamma\|$ , is in  $\mathcal{C}$ , then we say that  $\text{SHAP}(\mathcal{Q}, \text{PR}, \mathfrak{s})$  is in  $\mathcal{C}$  in combined complexity.
- If for every fixed  $Q \in \mathcal{Q}$ , the complexity of solving  $\text{SHAP}(\{Q\}, \text{PR}, \mathfrak{s})$ , as a function of  $\|\mathbf{p}^*\| + \|D\| + \|\Gamma\|$ , is in  $\mathcal{C}$ , then we say that  $\text{SHAP}(\mathcal{Q}, \text{PR}, \mathfrak{s})$  is in  $\mathcal{C}$  in data complexity.

That is, we discuss the computational complexity in terms of variants of the classical combined, and data complexity [34]. If not indicated otherwise, our statements about the complexity of  $\text{SHAP}(\mathcal{Q}, \text{PR}, \mathfrak{s})$  always refer to *combined* complexity.

## 4 General Insights for Fully Factorized Distributions

We first discuss fully factorized distributions. We start by showing that for tractable similarity functions, tractability of  $\text{SHAP}(\mathcal{Q}, \text{IND}, \mathfrak{s})$  in terms of data complexity is guided by the data complexity of  $\mathcal{Q}$ . This contrasts other query evaluation settings involving probabilities, like query evaluation in probabilistic databases [30, 33]. The intuitive reason is that here, the probability distributions are tied to the parameterized query instead of the database.

► **Proposition 4.1.** Let  $\mathfrak{s}$  be a tractable similarity function and let  $Q(\mathbf{R}; \mathbf{P})$  be a fixed parameterized query such that  $Q_{\mathbf{p}}(D)$  can be computed in polynomial time in  $\|D\|$  for all  $\mathbf{p} \in \text{Typ}[\mathbf{P}]$ . Then  $\text{SHAP}(\{Q\}, \text{IND}, \mathfrak{s})$  can be solved in polynomial time.

**Proof sketch.** When the parameterized query is fixed,  $\ell$  is constant. In particular, the probability spaces  $\Pi_i$  are of constant size. The problem can thus be solved by brute force, i.e., by evaluating the explicit formula for the SHAP score. ◀

For this reason, in the remainder of the paper, we will focus on the *combined complexity* of the computation of  $\text{SHAP}(\mathcal{Q}, \text{IND}, \mathfrak{s})$ .

Next, we point out two relationships with other computational problems. First, we see that  $\text{SHAP}(\mathcal{Q}, \text{IND}, \mathfrak{s})$  is at least as hard as deciding whether the output of a parameterized query is non-empty. To be precise, let  $\mathcal{Q}$  be a class of parameterized queries and let  $\text{NONEMPTY}(\mathcal{Q})$  be the problem that takes inputs  $(Q, \mathbf{p}^*, D)$ , and asks to decide whether  $Q_{\mathbf{p}^*}(D) \neq \emptyset$ . In order to establish a relationship to the  $\text{SHAP}(\mathcal{Q}, \text{IND}, \mathfrak{s})$  problem, we need two more ingredients.

► **Definition 4.2.** A similarity function  $\mathfrak{s}$  strongly depends on its first argument, if for all possible range schemas  $\mathbf{R}$ , one of the following is satisfied:

- (s1) for all non-empty  $T \in \text{Rel}[\mathbf{R}]$  we have  $\mathfrak{s}_{\mathbf{R}}(\emptyset, \emptyset) \neq \mathfrak{s}_{\mathbf{R}}(T, \emptyset)$ ; or
- (s2) for all non-empty  $T \in \text{Rel}[\mathbf{R}]$  we have  $\mathfrak{s}_{\mathbf{R}}(\emptyset, T) \neq \mathfrak{s}_{\mathbf{R}}(T, T)$ .

Both are highly natural conditions for meaningful similarity functions. Moreover, they entail that  $\mathfrak{s}$  can, to a certain extent, distinguish whether its first argument is empty or not. For example, the similarity functions from Example 3.2(a)–(c) all satisfy (s2), whereas a version of  $\text{NegDiff}$  that swaps the roles of  $Q$  and  $Q^*$  would always satisfy (s1).

► **Theorem 4.3.** Let  $\mathfrak{s}$  strongly depend on its first argument. Let  $\mathcal{Q}$  be a class of parameterized queries such that for all  $Q(\mathbf{R}; \mathbf{P}) \in \mathcal{Q}$  there exists  $i_0 \in [\ell]$  (which we can find in polynomial time) such that, for all  $D \in \text{DB}[\text{dom}(Q)]$  and all  $\mathbf{p} = (p_1, \dots, p_\ell) \in \text{Typ}[\mathbf{P}]$  with  $p_{i_0} \notin \text{adom}(D)$ , we have  $Q_{\mathbf{p}}(D) = \emptyset$ . Then  $\text{NONEMPTY}(\mathcal{Q}) \leq_{\text{T}}^{\text{P}} \text{SHAP}(\mathcal{Q}, \text{IND}, \mathfrak{s})$ .

## 14:12 The Importance of Parameters in Database Queries

**Proof sketch.** Our probability distribution  $\Gamma$  is just a coin flip between two parameter values  $\mathbf{p}^*$  and  $\mathbf{p}^a$ , where  $\mathbf{p}^a$  coincides with  $\mathbf{p}^*$  on all parameters except for  $p_{i_0}^*$  which is replaced with a value  $a \neq p_{i_0}^*$  outside of the active domain of  $D$ . Calculation shows that the  $\text{SHAP}(i_0) = 0$  if and only if  $Q_{\mathbf{p}^*}(D) = \emptyset$ . (We reduce to the SHAP instance  $(Q, \mathbf{p}^a, D, \Gamma)$  or  $(Q, \mathbf{p}^*, D, \Gamma)$ , depending on whether  $\mathfrak{s}$  satisfies (s1) or (s2)). ◀

The restriction we impose on  $\mathcal{Q}$  expresses that one of the parameters immediately renders the query result on  $D$  empty, if it is chosen outside the active domain of  $D$ . This is fulfilled, for example, for unions of conjunctive queries for which there exists a parameter that appears in each of its conjunctive queries, and can be extended to Datalog queries for which there exists a parameter that appears in the body of every rule.

Van den Broeck et al. [32] have shown a reduction from computing SHAP to computing the expected similarity. The latter problem,  $\text{ESIM}(\mathcal{Q}, \text{PR}, \mathfrak{s})$ , takes the same parameters and inputs as  $\text{SHAP}(\mathcal{Q}, \text{PR}, \mathfrak{s})$ , but asks for  $\mathbb{E}_{\mathbf{p} \sim \Gamma}[\mathfrak{s}(\mathbf{p}, \mathbf{p}^*)]$  instead of  $\text{SHAP}(i)$ .

► **Theorem 4.4** (Cf. [32, Theorem 2]). *For any class of parameterized queries  $\mathcal{Q}$  and any similarity function  $\mathfrak{s}$  such that  $\mathfrak{s} \circ \mathcal{Q}$  is tractable, we have  $\text{SHAP}(\mathcal{Q}, \text{IND}, \mathfrak{s}) \equiv_{\top}^{\text{P}} \text{ESIM}(\mathcal{Q}, \text{IND}, \mathfrak{s})$ .*

► **Remark 4.5.** The paper [32] discusses a more general setting, in which SHAP scores are computed for tractable functions  $F$  over  $\ell$ -ary domains. They provide an algorithm that computes SHAP scores of  $F$  using oracle calls to the expected value of  $F$  with different parameter distributions  $\Gamma \in \text{IND}$ . This algorithm runs in polynomial time in  $\|\Gamma\|$  and is independent of the choice of  $F$  (apart from the oracle, of course). For our version of the theorem, we use this algorithm on functions  $F_{Q, \mathbf{p}^*, D}(\mathbf{p}) = \mathfrak{s}(Q_{\mathbf{p}}(D), Q_{\mathbf{p}^*}(D))$ . ◻

The key property we needed for Proposition 4.1 was the manageable size of the parameter distributions. This can be formulated as a property of a class of queries.

► **Definition 4.6.** *Let  $Q(\mathbf{R}; \mathbf{P})$  be a parameterized query and  $D \in \text{DB}[\text{dom}(Q)]$ , and denote  $\text{psupp}(Q, D) := \{\mathbf{p} \in \text{Tup}[\mathbf{P}]: Q_{\mathbf{p}}(D) \neq \emptyset\}$ . A class  $\mathcal{Q}$  of parameterized queries has polynomially computable parameter support if there exists a polynomial time algorithm (in  $\|Q\| + \|D\|$ ) which, on input  $Q(\mathbf{R}; \mathbf{P}) \in \mathcal{Q}$  and  $D \in \text{DB}[\text{dom}(Q)]$ , outputs a set  $P(Q, D) \supseteq \text{psupp}(Q, D)$ .*

There are simple, yet relevant classes with this property, e.g., parameterized CQs with a bounded number of joins, or where parameters only appear in a bounded number of atoms. This property allows the efficient computation of expected similarities.

► **Proposition 4.7.** *Let  $\mathcal{Q}$  have polynomially computable parameter support, and suppose  $\mathfrak{s} \circ \mathcal{Q}$  is tractable. Then  $\text{SHAP}(\mathcal{Q}, \text{IND}, \mathfrak{s})$  can be solved in polynomial time.*

**Proof sketch.** We solve  $\text{ESIM}(\mathcal{Q}, \text{IND}, \mathfrak{s})$  efficiently, by first computing some  $P \supseteq \text{psupp}(Q, D)$  and then evaluating the formula for  $\mathbb{E}_{\mathbf{p}}[\mathfrak{s}(\mathbf{p}, \mathbf{p}^*)]$ . For this, we explicitly compute the terms for  $\mathbf{p} \in P$ , and bundle those for  $\mathbf{p} \notin P$ . By Theorem 4.4, this yields tractability of  $\text{SHAP}(\mathcal{Q}, \text{IND}, \mathfrak{s})$ . ◀

## 5 Parameterized Conjunctive Queries with Independent Parameters

In this section, we study the complexity of the SHAP score for the parameters of acyclic conjunctive queries. Later in the section, we also investigate the addition of inequalities to such queries.

## 5.1 Acyclic Conjunctive Queries

We first focus on parameterized *acyclic* conjunctive queries (pACQs). In particular, we will also consider classes of Boolean parameterized queries. If  $\mathbf{R}$  is a Boolean relation schema, then the only possible inputs to a similarity function  $\mathfrak{s}_{\mathbf{R}}$  are pairs of **tt** and **ff**. In particular, the properties from Definition 4.2 become

(s1')  $\mathfrak{s}_{\mathbf{R}}(\mathbf{ff}, \mathbf{ff}) \neq \mathfrak{s}_{\mathbf{R}}(\mathbf{tt}, \mathbf{ff})$  for (s1);

(s2')  $\mathfrak{s}_{\mathbf{R}}(\mathbf{ff}, \mathbf{tt}) \neq \mathfrak{s}_{\mathbf{R}}(\mathbf{tt}, \mathbf{tt})$  for (s2).

If  $\mathfrak{s}$  takes Boolean inputs and is strongly dependent on its first argument, then  $\mathfrak{s}$  satisfies one of (s1'), (s2'). Such  $\mathfrak{s}$  is also always tractable, because it has only four possible inputs: the pairs of the constants **tt** and **ff**.

► **Proposition 5.1.** *Let  $\mathcal{Q}$  be the class of Boolean parameterized queries of the shape*

$$Q(; y_1, \dots, y_\ell) = \exists x: R_1(x, y_1) \wedge \dots \wedge R_\ell(x, y_\ell) \quad (4)$$

and let  $\mathfrak{s}$  be strongly dependent on its first argument. Then  $\text{SHAP}(\mathcal{Q}, \text{IND}, \mathfrak{s})$  is #P-hard.

**Proof sketch.** It can easily be checked that  $\mathcal{Q}$  is tractable (although it does not have polynomially computable parameter support). Therefore, so is  $\mathfrak{s} \circ \mathcal{Q}$ . We use Theorem 4.4 and show that  $\text{ESIM}(\mathcal{Q}, \text{IND}, \mathfrak{s})$  is #P-hard by reduction from #posDNF. The main idea is as follows. Let  $\varphi$  be a positive DNF formula with variables  $X_i$  and disjuncts  $\varphi_j$ . Construct a database  $D$  in which the tuples  $R_i(j, y_i)$  represent the possible truth assignments for  $X_i$  to satisfy  $\varphi_j$ : If  $X_i$  occurs in  $\varphi_j$ , then  $y_i$  should be **tt**; otherwise,  $X_i$  does not matter in  $\varphi_j$  and  $y_i$  can take both values **tt** or **ff**. With this interpretation,  $Q$ , on  $D$ , expresses the existence of a disjunct  $\varphi_j$ , in which all occurring variables are set to **tt**. The expected similarity under the uniform distribution can be used to recover # $\varphi$ . ◀

► **Remark 5.2.** Our hardness results are stated for  $\text{PR} = \text{IND}$ . Inspection of our proofs reveals that they already hold for the subclass  $\text{PR} = \text{UNIF}$  of parameter distributions that are uniform on their support. ▽

Next, we show a tractability result for parameterized *full* ACQs. In contrast, by Proposition 5.1, even a single existential quantifier can make the problem difficult. A similar observation has been made for the (weighted) counting for ACQ answers [24, 9] (which we use to establish tractability here).

► **Proposition 5.3.** *Let  $\mathcal{Q}$  be the class of full pACQs, and let  $\mathfrak{s}$  be any of  $\text{Int}$ ,  $\text{NegSymDiff}$ , or  $\text{NegDiff}$ . Then  $\text{SHAP}(\mathcal{Q}, \text{IND}, \mathfrak{s})$  can be solved in polynomial time.*

**Proof sketch.** The main idea of the proof is to introduce an artificial “similarity” function  $\text{Count}$  with  $\text{Count}(T_1, T_2) = |T_1|$ . Tractability of  $\text{Count} \circ \mathcal{Q}$  is given, since counting the answers to full ACQs is tractable [24]. Thus, we can use Theorem 4.4 again and discuss  $\text{ESIM}(\mathcal{Q}, \text{IND}, \text{Count})$  first. This problem can be reduced to the weighted answer counting problem for ACQs, which is also tractable for full ACQs by [9]. A simple construction allows transferring tractability to  $\text{SHAP}(\mathcal{Q}, \text{IND}, \text{Int})$ . The similarities  $\text{NegSymDiff}$  and  $\text{NegDiff}$  are linear combinations in  $\text{Int}$  and  $\text{Count}$ . Therefore, we can compute the SHAP scores for the former two efficiently by computing the SHAP scores for the latter two. ◀

► **Remark 5.4.** In light of [9, Proposition 5], our tractability result for  $\text{Int}$  may seem surprising. The construction in our proof relies on the fact that in our case, the intersection concerns different parameterizations of the same query, one of them being fixed. If it were performed for two arbitrary full ACQs, acyclicity can be lost. ▽

## 5.2 Conjunctive Queries with Inequalities

In this section, we investigate parameterized conjunctive queries  $Q$  with inequalities of the shape  $x_i \leq y_j$ , where  $x_i$  is a variable (which also appears in the inequality-free part of  $Q$ ) and  $y_j$  is a parameter. We refer to such queries as  $p^{\leq}$ CQs. For simplicity, we assume that all attribute domains are numerical.

If  $\mathcal{Q}$  is a class of pCQs (i.e., without inequalities), then for  $t = 0, 1, 2, \dots$  we let  $\mathcal{Q}^{(\leq, t)}$  denote the class of  $p^{\leq}$ CQs obtained by adding at most  $t$  inequalities of the above shape to the body of a query from  $\mathcal{Q}$ . We let  $\mathcal{Q}^{\leq} = \bigcup_{t=0}^{\infty} \mathcal{Q}^{(\leq, t)}$ .

► **Remark 5.5.** In practice, such inequalities would typically be used for attributes with continuous domains, like  $\mathbb{R}$ . Continuous parameter values would require a treatment of continuous parameter distributions. Our framework can handle continuous parameter distributions as follows: On the technical side, the definition of the SHAP score needs to be changed to avoid conditional probabilities. For fully factorized distributions, this can be done easily. On the algorithmic side, we can discretize the continuous distribution into intervals defined by the values in the database to obtain a discrete distribution with finite support that yields the same output. ◀

► **Proposition 5.6.** *Let  $\mathcal{Q}$  be the class of  $p^{\leq}$ CQs of the shape*

$$Q(; y_1, \dots, y_\ell) = \exists x_1, \dots, x_\ell: R(x_1, \dots, x_\ell) \wedge (x_1 \leq y_1) \wedge \dots \wedge (x_\ell \leq y_\ell). \quad (5)$$

*and let  $\mathfrak{s}$  be strongly dependent on its first argument. Then  $\text{SHAP}(\mathcal{Q}, \text{IND}, \mathfrak{s})$  is  $\#P$ -hard.*

**Proof sketch.** This proof works very similar to that of Proposition 5.1. First, we observe that  $\mathcal{Q}$  is tractable (but again not with polynomially parameter support). Then  $\mathfrak{s} \circ \mathcal{Q}$  is tractable too. In the remainder of the proof, we prove hardness of  $\text{ESIM}(\mathcal{Q}, \text{IND}, \mathfrak{s})$  by reduction from  $\#\text{posDNF}$ , similar to the proof of Proposition 5.1. ◀

The simple shape of (5) indicates that structural restrictions on the usage of inequalities are needed to establish tractability. If  $\mathcal{Q}$  is a class of pCQs (i.e., without  $\leq$ ), then we call a query  $Q \in \mathcal{Q}^{\leq}$  *acyclic* (a  $p^{\leq}$ ACQ), if it becomes a pACQ, when  $\leq$  is interpreted as a relation symbol. It can be shown that Proposition 5.3 extends to this setting:

► **Proposition 5.7.** *Let  $\mathcal{Q}$  be the class of full pACQs (i.e., without  $\leq$ ), and let  $\mathcal{Q}'$  be a class of full  $p^{\leq}$ ACQs such that  $\mathcal{Q}' \subseteq \mathcal{Q}^{\leq}$ . Moreover, let  $\mathfrak{s}$  be any of  $\text{Int}$ ,  $\text{NegSymDiff}$ , or  $\text{NegDiff}$ . Then  $\text{SHAP}(\mathcal{Q}', \text{IND}, \mathfrak{s})$  can be computed in polynomial time.*

**Proof sketch.** We replace inequalities  $x_i \leq y_j$  with atoms  $R_{\leq}(x_i, y_j)$  and hard-code the relevant  $R_{\leq}$  tuples in the database  $D$ . Then, Proposition 5.3 can be applied. ◀

## 6 Correlated Parameters and Approximability

In this section, we allow classes  $\text{PR}$  of parameter distributions with correlations. We only need to make the following tractability assumptions (which are trivially satisfied by  $\text{IND}$ ).

1. For every fixed  $\ell$ , and every  $\Gamma \in \text{PR}_{\mathcal{P}}$  with  $|\mathcal{P}| = \ell$ , the support  $\text{supp}(\Gamma)$  can be computed in polynomial time in  $\|\Gamma\|$ .
2. For all  $\mathbf{p}$  and  $J$ , we can compute  $\Pr_{\mathbf{p}' \sim \Gamma}(\mathbf{p}'_J = \mathbf{p}_J)$  in polynomial time in  $\|\Gamma\|$ .

For example, the first property holds for distributions encoded by Bayesian networks. The second one holds for structurally restricted classes of Bayesian networks, like polytrees [13]. The following result states that given these assumptions, the data complexity of the SHAP problem remains in polynomial time even for parameter distributions with correlations. The proof is similar to the proof of Proposition 4.1.

► **Proposition 6.1.** *Let  $\mathfrak{s}$  be a tractable similarity function and let  $Q(\mathbf{R}; \mathbf{P})$  be a fixed parameterized query such that  $Q_{\mathbf{p}}(D)$  can be computed in polynomial time in  $\|D\|$  for all  $\mathbf{p} \in \text{Tup}[\mathbf{P}]$ . Moreover, let  $\text{PR}$  be a class of distributions as described above. Then  $\text{SHAP}(\{Q\}, \text{PR}, \mathfrak{s})$  can be solved in polynomial time.*

We conclude this section by explaining how SHAP can be approximated via sampling. Consider an input  $(Q, \mathbf{p}^*, D, \Gamma)$  of  $\text{SHAP}(Q, \text{PR}, \mathfrak{s})$ , where  $Q = Q(\mathbf{R}; \mathbf{P})$  and  $|\mathbf{p}^*| = \ell$ . We rewrite  $\text{SHAP}(i)$  as an expectation in a single probability space, instead of nested expectations in different spaces. Let  $\{i\}^0 = \emptyset$  and  $\{i\}^1 = \{i\}$ . Consider the following two-step random process, for  $b \in \{0, 1\}$ :

1. Draw  $J \subseteq [\ell] \setminus i$  according to  $\Pi_i$ .
2. Draw  $\mathbf{p}$  according to  $\Gamma$ , conditioned on having  $\mathbf{p}$  agree with  $\mathbf{p}^*$  on  $J$ .

This defines a joint probability distribution on pairs  $(\mathbf{p}, J)$ . By  $\Gamma^{i,1}$  and  $\Gamma^{i,0}$ , we denote the corresponding marginal distributions over parameter tuples  $\mathbf{p}$ . A simple calculation shows that the following equality holds:

$$\text{SHAP}(i) = \mathbb{E}_{\mathbf{p} \sim \Gamma^{i,1}}[\mathfrak{s}(\mathbf{p}, \mathbf{p}^*)] - \mathbb{E}_{\mathbf{p} \sim \Gamma^{i,0}}[\mathfrak{s}(\mathbf{p}, \mathbf{p}^*)]. \quad (6)$$

We give a proof of this statement in the full version of the paper [11].

We say that  $\text{PR}$  *admits efficient conditional sampling* if for all  $\Gamma \in \text{PR}$ , all  $\mathbf{p}^* \in \text{supp}(\Gamma)$ , and all  $J \subseteq [\ell]$ , the conditional distribution of  $\Gamma$  subject to  $\mathbf{p}_J = \mathbf{p}_J^*$  can be sampled in polynomial time in  $\|\Gamma\| + \|\mathbf{p}^*\|$ . This is again the case, for example, for structurally restricted classes of Bayesian networks. By the structure of the two-step process defining  $\Gamma^{i,b}$ , we can efficiently sample from  $\Gamma^{i,b}$  if we can efficiently sample from conditional distributions of  $\Gamma$ . A proof of this can be found in the full version of the paper [11].

A similarity function  $\mathfrak{s}$  is *bounded* for a class of parameterized queries  $\mathcal{Q}$  if there are  $a \leq b$  such that for all  $(Q, \mathbf{p}^1, \mathbf{p}^2)$ ,  $Q \in \mathcal{Q}$ , we have  $a \leq \mathfrak{s}(\mathbf{p}^1, \mathbf{p}^2) \leq b$ . For example, similarity measures, like *Jaccard*, are usually  $[0, 1]$ -valued. The following theorem states that, under the assumptions of efficient conditional sampling and boundedness, we have an additive FPRAS for the SHAP score of a parameter.

► **Theorem 6.2.** *Let  $\mathcal{Q}$  be a class of tractable parameterized queries, let  $\mathfrak{s}$  a tractable similarity function, and let  $\text{PR}$  be a class of parameter distributions such that*

- (a) *Every  $\Gamma \in \text{PR}$  admits efficient conditional sampling.*
- (b) *The value range of  $\mathfrak{s}$  is bounded for  $\mathcal{Q}$ .*

*Then, for all inputs  $(Q, \mathbf{p}^*, D, \Gamma)$  and all  $i \in [\ell]$ , we can compute a value  $S$  satisfying  $\Pr(|S - \text{SHAP}(i)| < \varepsilon) \geq 1 - \delta$  in time polynomial in  $\frac{1}{\varepsilon}$ ,  $\log \frac{1}{\delta}$ , and the size of the input.*

**Proof sketch.** The proof uses a simple sampling procedure and approximates the two terms in Equation (6) from the means of the samples. The guarantee is then given by a variant of Hoeffding's inequality (requiring boundedness of  $\mathfrak{s}$  to be applicable). ◀

## 7 Conclusions

We proposed a framework for measuring the responsibility of parameters to the result of a query using the SHAP score. We studied the computational problem of calculating the SHAP score of a given parameter value. We gave general complexity lower and upper bounds, and presented a complexity analysis for the restricted case of conjunctive queries and independent parameters. We also discussed the complexity of approximate calculation and correlated parameters. The rich framework we introduced here offers many opportunities for future

research. Especially important is the direction of aggregate queries, where the similarity between results accounts for the numerical values such as sum, average, median, and so on. For such queries, it is important to study numerical parameter distributions, which are typically continuous probability measures. It is also important to identify general tractability conditions for similarity measures and parameter distributions in order to generalize the upper bounds beyond the special cases that we covered here. Finally, we plan to explore the applicability of SHAP to measuring parameters in various queries and datasets, such as those studied in the context of fact checking [5, 31].

---

## References

- 1 Encarnación Algaba, Vito Fragnelli, and Joaquín Sánchez-Soriano, editors. *Handbook of the Shapley Value*. CRC Press, 2019. doi:10.1201/9781351241410.
- 2 Dana Arad, Daniel Deutch, and Nave Frost. LearnShapley: Learning to predict rankings of facts contribution based on query logs. In *CIKM*, pages 4788–4792. ACM, 2022. doi:10.1145/3511808.3557204.
- 3 Marcelo Arenas, Pablo Barceló, Leopoldo E. Bertossi, and Mikaël Monet. The tractability of shap-score-based explanations for classification over deterministic and decomposable boolean circuits. In *AAAI*, pages 6670–6678. AAAI Press, 2021. doi:10.1609/aaai.v35i8.16825.
- 4 Leopoldo E. Bertossi, Loreto Bravo, Enrico Franconi, and Andrei Lopatenko. The complexity and approximation of fixing numerical attributes in databases under integrity constraints. *Inf. Syst.*, 33(4-5):407–434, 2008. doi:10.1016/j.is.2008.01.005.
- 5 Adriane Chapman and H. V. Jagadish. Why not? In *SIGMOD Conference*, pages 523–534. ACM, 2009. doi:10.1145/1559845.1559901.
- 6 Xiaotie Deng and Christos H. Papadimitriou. On the complexity of cooperative solution concepts. *Math. Oper. Res.*, 19(2):257–266, 1994. doi:10.1287/moor.19.2.257.
- 7 Daniel Deutch, Nave Frost, Amir Gilad, and Oren Sheffer. Explanations for data repair through shapley values. In *CIKM*, pages 362–371. ACM, 2021. doi:10.1145/3459637.3482341.
- 8 Daniel Deutch, Nave Frost, Benny Kimelfeld, and Mikaël Monet. Computing the shapley value of facts in query answering. In *SIGMOD Conference*, pages 1570–1583. ACM, 2022. doi:10.1145/3514221.3517912.
- 9 Arnaud Durand and Stefan Mengel. The complexity of weighted counting for acyclic conjunctive queries. *J. Comput. Syst. Sci.*, 80(1):277–296, 2014. doi:10.1016/j.jcss.2013.08.001.
- 10 U. Faigle and W. Kern. The shapley value for cooperative games under precedence constraints. *Int. J. Game Theory*, 21(3):249–266, sep 1992. doi:10.1007/BF01258278.
- 11 Martin Grohe, Benny Kimelfeld, Peter Lindner, and Christoph Standke. The importance of parameters in database queries, 2024. arXiv:2401.04606 [cs.DB]. doi:10.48550/arXiv.2401.04606.
- 12 Yuri Gurevich and Saharon Shelah. Time polynomial in input or output. *J. Symb. Log.*, 54(3):1083–1088, 1989. doi:10.2307/2274767.
- 13 Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, 2009.
- 14 Nick Koudas, Chen Li, Anthony K. H. Tung, and Rares Vernica. Relaxing join and selection queries. In *VLDB*, pages 199–210. ACM, 2006. URL: <http://dl.acm.org/citation.cfm?id=1164146>.
- 15 Marie-Jeanne Lesot, Maria Rifqi, and Hamid Benhadda. Similarity measures for binary and numerical data: a survey. *International Journal of Knowledge Engineering and Soft Data Paradigms*, 1(1):63–84, dec 2008. doi:10.1504/ijkesdp.2009.021985.
- 16 Yin Lin, Brit Youngmann, Yuval Moskovitch, H. V. Jagadish, and Tova Milo. On detecting cherry-picked generalizations. *Proc. VLDB Endow.*, 15(1):59–71, 2021. doi:10.14778/3485450.3485457.




- 17 Ester Livshits, Leopoldo E. Bertossi, Benny Kimelfeld, and Moshe Sebag. The Shapley value of tuples in query answering. In *ICDT*, volume 155 of *LIPIcs*, pages 20: 1–20: 19. Schloss Dagstuhl, 2020. doi:10.4230/LIPIcs.ICDT.2020.20.
- 18 Ester Livshits and Benny Kimelfeld. The shapley value of inconsistency measures for functional dependencies. *Log. Methods Comput. Sci.*, 18(2), 2022. doi:10.46298/lmcs-18(2:20)2022.
- 19 Scott M. Lundberg, Gabriel G. Erion, Hugh Chen, Alex J. DeGrave, Jordan M. Prutkin, Bala Nair, Ronit Katz, Jonathan Himmelfarb, Nisha Bansal, and Su-In Lee. From local explanations to global understanding with explainable AI for trees. *Nat. Mach. Intell.*, 2(1):56–67, 2020. doi:10.1038/s42256-019-0138-9.
- 20 Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *NIPS*, pages 4765–4774, 2017. URL: <https://proceedings.neurips.cc/paper/2017/hash/8a20a8621978632d76c43dfd28b67767-Abstract.html>.
- 21 Christoph Molnar. Interpretable machine learning: A guide for making black box models explainable, 2023. Version 2023-08-21. URL: <https://christophm.github.io/interpretable-ml-book>.
- 22 Davide Mottin, Alice Marascu, Senjuti Basu Roy, Gautam Das, Themis Palpanas, and Yannis Velegrakis. A probabilistic optimization framework for the empty-answer problem. *Proc. VLDB Endow.*, 6(14):1762–1773, 2013. doi:10.14778/2556549.2556560.
- 23 Santiago Ontañón. An overview of distance and similarity functions for structured data. *Artif. Intell. Rev.*, 53(7):5309–5351, 2020. doi:10.1007/s10462-020-09821-w.
- 24 Reinhard Pichler and Sebastian Skritek. Tractable counting of the answers to conjunctive queries. *J. Comput. Syst. Sci.*, 79(6):984–1001, 2013. doi:10.1016/j.jcss.2013.01.012.
- 25 Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “Why should I trust you?”: Explaining the predictions of any classifier. In *KDD*, pages 1135–1144. ACM, 2016. doi:10.1145/2939672.2939778.
- 26 Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-precision model-agnostic explanations. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018. doi:10.1609/aaai.v32i1.11491.
- 27 Alvin E. Roth. *The Shapley value: essays in honor of Lloyd S. Shapley*. Cambridge University Press, 1988.
- 28 B. Sathiya and T. V. Geetha. A review on semantic similarity measures for ontology. *J. Intell. Fuzzy Syst.*, 36(4):3045–3059, 2019. doi:10.3233/JIFS-18120.
- 29 Lloyd S. Shapley. A value for  $n$ -person games. In Harold W. Kuhn and Albert W. Tucker, editors, *Contributions to the Theory of Games II*, pages 307–317. Princeton University Press, Princeton, 1953.
- 30 Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. *Probabilistic Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011. doi:10.2200/S00362ED1V01Y201105DTM016.
- 31 Quoc Trung Tran and Chee-Yong Chan. How to conquer why-not questions. In *SIGMOD Conference*, pages 15–26. ACM, 2010. doi:10.1145/1807167.1807172.
- 32 Guy Van den Broeck, Anton Lykov, Maximilian Schleich, and Dan Suciu. On the tractability of SHAP explanations. *Journal of Artificial Intelligence Research*, 74:851–886, jun 2022. doi:10.1613/jair.1.13283.
- 33 Guy Van den Broeck and Dan Suciu. Query processing on probabilistic data: A survey. *Found. Trends Databases*, 7(3-4):197–341, 2017. doi:10.1561/19000000052.
- 34 Moshe Y. Vardi. The complexity of relational query languages (extended abstract). In Harry R. Lewis, Barbara B. Simons, Walter A. Burkhard, and Lawrence H. Landweber, editors, *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, pages 137–146. ACM, 1982. doi:10.1145/800070.802186.
- 35 You Wu, Pankaj K. Agarwal, Chengkai Li, Jun Yang, and Cong Yu. Computational fact checking through query perturbations. *ACM Trans. Database Syst.*, 42(1):4:1–4:41, 2017. doi:10.1145/2996453.









# Conjunctive Queries on Probabilistic Graphs: The Limits of Approximability

Antoine Amarilli   

LTCI, Télécom Paris, Institut Polytechnique de Paris, France

Timothy van Bremen  

National University of Singapore, Singapore

Kuldeep S. Meel  

University of Toronto, Canada

---

## Abstract

Query evaluation over probabilistic databases is a notoriously intractable problem – not only in combined complexity, but for many natural queries in data complexity as well [7, 14]. This motivates the study of probabilistic query evaluation through the lens of approximation algorithms, and particularly of *combined FPRASes*, whose runtime is polynomial in both the query and instance size. In this paper, we focus on tuple-independent probabilistic databases over binary signatures, which can be equivalently viewed as *probabilistic graphs*. We study in which cases we can devise combined FPRASes for probabilistic query evaluation in this setting.

We settle the complexity of this problem for a variety of query and instance classes, by proving both approximability and (conditional) inapproximability results. This allows us to deduce many corollaries of possible independent interest. For example, we show how the results of [8] on counting fixed-length strings accepted by an NFA imply the existence of an FPRAS for the two-terminal network reliability problem on directed acyclic graphs: this was an open problem until now [37]. We also show that one cannot extend a recent result [34] that gives a combined FPRAS for self-join-free conjunctive queries of bounded hypertree width on probabilistic databases: neither the bounded-hypertree-width condition nor the self-join-freeness hypothesis can be relaxed. Finally, we complement all our inapproximability results with unconditional lower bounds, showing that DNNF provenance circuits must have at least moderately exponential size in combined complexity.

**2012 ACM Subject Classification** Theory of computation → Database query processing and optimization (theory)

**Keywords and phrases** Probabilistic query evaluation, tuple-independent databases, approximation

**Digital Object Identifier** 10.4230/LIPIcs.ICDT.2024.15

**Funding** This project was supported in part by the National Research Foundation Singapore under its NRF Fellowship programme [NRF-NRFFAI1-2019-0004] and Campus for Research Excellence and Technological Enterprise (CREATE) programme, as well as the Ministry of Education Singapore Tier 1 and 2 grants R-252-000-B59-114 and MOE-T2EP20121-0011. Amarilli was partially supported by the ANR project EQUUS ANR-19-CE48-0019, by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 431183758, and by the ANR project ANR-18-CE23-0003-02 (“CQFD”). This work was done in part while Amarilli was visiting the Simons Institute for the Theory of Computing.

**Acknowledgements** The authors thank Octave Gaspard for pointing out some oversights in some proofs, which are corrected in the present version.

## 1 Introduction

*Tuple-independent probabilistic databases* (TID) are a simple and principled formalism to model uncertainty and noise in relational data [13, 32]. In the TID model, each tuple of a relational database is annotated with an independent probability of existence; all tuples are



© Antoine Amarilli, Timothy van Bremen, and Kuldeep S. Meel;  
licensed under Creative Commons License CC-BY 4.0

27th International Conference on Database Theory (ICDT 2024).

Editors: Graham Cormode and Michael Shekelyan; Article No. 15; pp. 15:1–15:20

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

assumed to be independent. In the *probabilistic query evaluation* (PQE) problem, given a Boolean query  $Q$  and a TID instance  $I$ , we must compute the probability that  $Q$  holds in a subinstance sampled from  $I$  according to the resulting distribution. The PQE problem has been studied in database theory both in terms of *combined complexity*, where the query and instance are part of the input, and in *data complexity*, where the query is fixed and only the instance is given as input [35]. Unfortunately, many of the results so far [32] show that the PQE problem is highly intractable, even in data complexity for many natural queries (e.g., a path query of length three), and hence also in combined complexity.

Faced with this intractability, a natural approach is to study *approximate PQE*: we relax the requirement of computing the exact probability that the query holds, and settle for an approximate answer. This approach has been studied in data complexity [32]: for any fixed union of conjunctive queries (UCQ), we can always tractably approximate the answer to PQE, additively (simply by Monte Carlo sampling), or multiplicatively (using the Karp-Luby approximation algorithm on a disjunctive-normal-form representation of the query provenance). However, these approaches are not tractable in combined complexity, and moreover the latter approach exhibits a “slicewise polynomial” runtime of the form  $O(|I|^{|Q|})$  – rather than, say,  $O(2^{|Q|} \text{poly}(|I|))$  – which seriously limits its practical utility. Thus, our goal is to obtain a *combined FPRAS* for PQE: by this we mean a fully polynomial-time randomized approximation scheme, giving a multiplicative approximation of the probability, whose runtime is polynomial in the query and TID (and in the desired precision). This approach has been recently proposed by van Bremen and Meel [34], who show a combined FPRAS for CQs when assuming that the query is self-join-free and has bounded hypertree width; their work leaves open the question of which other cases admit combined FPRASes.

**Main Results.** In this paper, following the work of Amarilli, Monet and Senellart [7] for exact PQE, we investigate the combined complexity of *approximate PQE* in the setting of *probabilistic graphs*. In other words, we study *probabilistic graph homomorphism*, which is the equivalent analogue of CQ evaluation: given a (deterministic) query graph  $G$ , and given a instance graph  $H$  with edges annotated with independent probabilities (like a TID), we wish to approximate the probability that a randomly selected subgraph  $H' \subseteq H$  admits a homomorphism from  $G$ . This setting is incomparable to that of [34], because it allows for self-joins and for queries of unbounded width, but assumes that relations are binary.

Of course, the graph homomorphism problem is intractable in combined complexity if the input graphs are arbitrary (even without probabilities). Hence, we study the problem when the query graph and instance graph are required to fall in restricted graph classes, chosen to ensure tractability in the non-probabilistic setting. We use similar classes as those from [7]: *path graphs* which may be *one-way* (1WP: all edges are oriented from left to right) or *two-way* (2WP: edge orientations are arbitrary); *tree graphs* which may be *downward* (DWT: all edges are oriented from the root to the leaves) or *polytrees* (PT: edge orientations are arbitrary); and, for the instance graph, *directed acyclic graphs* (DAG), or *arbitrary graphs* (All).

For all combinations of these classes, we show either (i) the existence of a combined FPRAS, or (ii) the non-existence of such an FPRAS, subject to standard complexity-theoretic assumptions. We summarize our results in Table 1, respectively for graphs that are *labelled* (i.e., the signature features several binary relations), or *unlabelled* (i.e., only one binary relation). We emphasize that the signature for labelled graphs is assumed to be fixed and does not form part of the input, consistent with prior work [7] (although identical results can likely be obtained even when dropping this assumption).

► **Result 1.1** (Sections 3 and 4). *The results in Table 1, described in terms of the graph classes outlined above, hold.*

In summary, for the classes that we consider, our results mostly show that the general intractability of combined PQE carries over to the approximate PQE problem. The important exception is Proposition 3.1: the PQE problem for one-way path queries on *directed acyclic graphs* (DAGs) admits a combined FPRAS. We discuss more in detail below how this result is proved and some of its consequences. Another case is left open: in the unlabelled setting, we do not settle the approximability of combined PQE for one-way path queries (or equivalently downward tree queries) on arbitrary graphs. For all other cases, either exact combined PQE was already shown to be tractable in the exact setting [7], or we strengthen the #P-hardness of exact PQE from [7] by showing that combined FPRASes conditionally do not exist. We stress that our results always concern *multiplicative approximations*: as non-probabilistic graph homomorphism is tractable for the classes that we consider, we can always obtain additive approximations for PQE simply by Monte Carlo sampling. Further note that our intractability results are always shown in *combined complexity* – in data complexity, for the queries that we consider, PQE is always multiplicatively approximable via the Karp-Luby algorithm [32].

As an important consequence, our techniques yield connections between approximate PQE and *intensional* approaches to the PQE problem. Recall that the intensional approach was introduced by Jha and Suciu [21] in the setting of exact evaluation, and when measuring data complexity. They show that many tractable queries for PQE also admit tractable provenance representations. More precisely, for these queries  $Q$ , there is a polynomial-time algorithm that takes as input any database instance and computes a representation of the Boolean provenance of  $Q$  in a form which admits tractable model counting (e.g., OBDD, d-DNNF, etc.). This intensional approach contrasts with *extensional* approaches (like [14]) which exploit the structure of the query directly: comparing both approaches is still open [27].

In line with this intensional approach, we complement our conditional hardness results on approximate PQE with *unconditional* lower bounds on the *combined* size of tractable representations of query provenance. Namely, we show a moderately exponential lower bound on DNNF provenance representations for all our non-approximable query-instance class pairs:

► **Result 1.2** (Section 5, informal). *Let  $(\mathcal{G}, \mathcal{H})$  be a conditionally non-approximable query-instance class pair studied in this paper. For any  $\epsilon > 0$ , there is an infinite family  $G_1, G_2, \dots$  of  $\mathcal{G}$  queries and an infinite family  $H_1, H_2, \dots$  of  $\mathcal{H}$  instances such that, for any  $i > 0$ , any DNNF circuit representing the provenance  $\text{Prov}_{H_i}^{G_i}$  has size at least  $2^{\Omega((\|G_i\| + \|H_i\|)^{1-\epsilon})}$ .*

The class of DNNF circuits is arguably the most succinct circuit class in knowledge compilation that still has desirable properties [15, 16]. Such circuits subsume in particular the class of *structured DNNFs*, for which tractable approximation algorithms were recently proposed [9]. Thus, these bounds help to better understand the limitations of intensional approaches.

**Consequences.** Our results and techniques have several interesting consequences of potential independent interest. First, they imply that we cannot relax the hypotheses of the result of van Bremen and Meel mentioned earlier [34]. They show the following result on combined FPRASes for PQE in the more general context of probabilistic databases:

► **Theorem 1.3** (Theorem 1 of [34]). *Let  $Q$  be a self-join-free conjunctive query of bounded hypertree width, and  $H$  a tuple-independent database instance. Then there exists a combined FPRAS for computing the probability of  $Q$  on  $H$ , i.e., an FPRAS whose runtime is  $\text{poly}(|Q|, \|H\|, \epsilon^{-1})$ , where  $\epsilon$  is the multiplicative error.*

It was left open in [34] whether intractability held without these assumptions on the query. Hardness is immediate if we do not bound the width of queries and allow arbitrary self-join-free CQs, as combined query evaluation is then NP-hard already in the non-probabilistic setting. However, it is less clear whether the self-join-freeness condition can be lifted. Our results give a negative answer, already in a severely restricted setting:

► **Result 1.4** (Corollaries 6.1 and 6.2). *Assuming  $RP \neq NP$ , neither the bounded hypertree width nor self-join-free condition in Theorem 1.3 can be relaxed: even on a fixed signature consisting of a single binary relation, there is no FPRAS to approximate the probability of an input treewidth-1 CQ on an input treewidth-1 TID instance.*

A second consequence implied by our techniques concerns the *two-terminal network reliability problem* on directed acyclic graphs (DAGs). Roughly speaking, given a directed graph  $G = (V, E)$  with independent edge reliability probabilities  $\pi : E \rightarrow [0, 1]$ , and two distinguished vertices  $s, t \in V$ , the two-terminal network reliability problem asks for the probability that there is a path from  $s$  to  $t$ . The problem is known to be #P-hard even on DAGs [29, Table 2]. The existence of an FPRAS for the two-terminal network reliability problem is a long-standing open question [22], and the case of DAGs was explicitly left open by Zenklusen and Laumanns [37]. Our results allow us to answer in the affirmative:

► **Result 1.5** (Theorem 6.3). *There exists an FPRAS for the two-terminal network reliability problem over DAGs.*

This result and our approximability results follow from the observation that path queries on directed acyclic graphs admit a compact representation of their Boolean provenance as *non-deterministic ordered binary decision diagrams* (nOBDDs). We are then able to use a recent result by Arenas et al. [8, Corollary 4.5] giving an FPRAS for counting the satisfying assignments of an nOBDD, adapted to the weighted setting.

**Paper Structure.** In Section 2, we review some of the technical background. We then present our main results on approximability, divided into the labelled and unlabelled case, in Sections 3 and 4 respectively. Next, in Section 5, we show lower bounds on DNNF provenance circuit sizes. In Section 6, we show some consequences for previous work [34], as well as for the two-terminal network reliability problem. We conclude in Section 7.

## 2 Preliminaries

We provide some technical background below, much of which closely follows that in [4] and [7].

**Graphs and Graph Homomorphisms.** Let  $\sigma$  be a non-empty finite set of labels. When  $|\sigma| > 1$ , we say that we are in the *labelled setting*, and when  $|\sigma| = 1$ , the *unlabelled setting*. In this paper, we study only *directed* graphs with edge labels from  $\sigma$ . A graph  $G$  over  $\sigma$  is a tuple  $(V, E, \lambda)$  with finite non-empty vertex set  $V$ , edge set  $E \subseteq V^2$ , and  $\lambda : E \rightarrow \sigma$  a labelling function mapping each edge to a single label (we may omit  $\lambda$  in the unlabelled setting). The *size*  $\|G\|$  of  $G$  is its number of edges. We write  $x \xrightarrow{R} y$  for an edge  $e = (x, y) \in E$  with label  $\lambda(e) = R$ , and  $x \rightarrow y$  for  $(x, y) \in E$  (no matter the edge label). We sometimes use a simple regular-expression-like syntax (omitting the vertex names) to represent path graphs: for example, we write  $\rightarrow\rightarrow$  to represent an unlabelled path of length two, and the notation  $\rightarrow^k$  to denote an unlabelled path of length  $k$ . All of this syntax extends to labelled graphs in the obvious way. A graph  $H = (V', E', \lambda')$  is a *subgraph* of  $G$ , written  $H \subseteq G$ , if  $V = V'$ ,  $E' \subseteq E$ , and  $\lambda'$  is the restriction of  $\lambda$  to  $E'$ .

A *graph homomorphism*  $h$  from a graph  $G = (V_G, E_G, \lambda_G)$  to a graph  $H = (V_H, E_H, \lambda_H)$  is a function  $h : V_G \rightarrow V_H$  such that, for all  $(u, v) \in E_G$ , we have  $(h(u), h(v)) \in E_H$  and  $\lambda_H((h(u), h(v))) = \lambda_G((u, v))$ . We write  $G \rightsquigarrow H$  to say that such a homomorphism exists.

**Probabilistic Graphs and Probabilistic Graph Homomorphism.** A *probabilistic graph* is a pair  $(H, \pi)$ , where  $H$  is a graph with edge labels from  $\sigma$ , and  $\pi : E \rightarrow [0, 1]$  is a probability labelling on the edges. Note that edges  $e$  in  $H$  are annotated both by their probability value  $\pi(e)$  and their  $\sigma$ -label  $\lambda(e)$ . Intuitively,  $\pi$  gives us a succinct specification of a probability distribution over the  $2^{|E|}$  possible subgraphs of  $H$ , by independently including each edge  $e$  with probability  $\pi(e)$ . Formally, the distribution induced by  $\pi$  on the subgraphs  $H' \subseteq H$  is defined by  $\Pr_\pi(H') = \prod_{e \in E'} \pi(e) \prod_{e \in E \setminus E'} (1 - \pi(e))$ .

In this paper, we study the *probabilistic graph homomorphism* problem PHom for a fixed set of labels  $\sigma$ : given a graph  $G$  called the *query graph* and a probabilistic graph  $(H, \pi)$  called the *instance graph*, both using labels from  $\sigma$ , we must compute the probability  $\Pr_\pi(G \rightsquigarrow H)$  that a subgraph of  $H$ , sampled according to the distribution induced by  $\pi$ , admits a homomorphism from  $G$ . That is, we must compute  $\Pr_\pi(G \rightsquigarrow H) := \sum_{H' \subseteq H \text{ s.t. } G \rightsquigarrow H'} \Pr_\pi(H')$ .

We study PHom in *combined complexity*, i.e., when both the query graph  $G$  and instance graph  $(H, \pi)$  are given as input. Further, we study PHom when we restrict  $G$  and  $H$  to be taken from specific *graph classes*, i.e., infinite families of (non-probabilistic) graphs, denoted respectively  $\mathcal{G}$  and  $\mathcal{H}$ . (Note that  $\mathcal{H}$  does not restrict the probability labelling  $\pi$ .) To distinguish the *labelled* and *unlabelled* setting, we denote by  $\text{PHom}_L(\mathcal{G}, \mathcal{H})$  the problem of computing  $\Pr_\pi(G \rightsquigarrow H)$  for  $G \in \mathcal{G}$  and  $(H, \pi)$  with  $H \in \mathcal{H}$  when the fixed set of allowed labels in  $\mathcal{G}$  and  $\mathcal{H}$  has cardinality  $|\sigma| > 1$ , and likewise write  $\text{PHom}_U(\mathcal{G}, \mathcal{H})$  when  $\mathcal{G}$  and  $\mathcal{H}$  are classes of unlabelled graphs. We focus on *approximation algorithms*: fixing classes  $\mathcal{G}$  and  $\mathcal{H}$ , a *fully polynomial-time randomized approximation scheme* (FPRAS) for  $\text{PHom}_L(\mathcal{G}, \mathcal{H})$  (in the labelled setting) or  $\text{PHom}_U(\mathcal{G}, \mathcal{H})$  (in the unlabelled setting) is a randomized algorithm that runs in time  $\text{poly}(\|G\|, \|H\|, \epsilon^{-1})$  on inputs  $G \in \mathcal{G}$ ,  $(H, \pi)$  for  $H \in \mathcal{H}$ , and  $\epsilon > 0$ . The algorithm must return, with probability at least  $3/4$ , a *multiplicative approximation* of the probability  $\Pr_\pi(G \rightsquigarrow H)$ , i.e., a value between  $(1 - \epsilon) \Pr_\pi(G \rightsquigarrow H)$  and  $(1 + \epsilon) \Pr_\pi(G \rightsquigarrow H)$ .

**Graph Classes.** We study PHom on the following graph classes, which are defined on a graph  $G$  with edge labels from  $\sigma$ , and are either labelled or unlabelled depending on  $\sigma$ :

- $G$  is a *one-way path* (1WP) if it is of the form  $a_1 \xrightarrow{R_1} \dots \xrightarrow{R_{m-1}} a_m$  for some  $m$ , with all  $a_1, \dots, a_m$  being pairwise distinct, and with  $R_i \in \sigma$  for  $1 \leq i < m$ .
- $G$  is a *two-way path* (2WP) if it is of the form  $a_1 - \dots - a_m$  for some  $m$ , with pairwise distinct  $a_1, \dots, a_m$ , and each  $-$  being  $\xrightarrow{R_i}$  or  $\xleftarrow{R_i}$  (but not both) for some label  $R_i \in \sigma$ .
- $G$  is a *downward tree* (DWT) if it is a rooted unranked tree (each node can have an arbitrary number of children), with all edges pointing from parent to child in the tree.
- $G$  is a *polytree* (PT) if its underlying undirected graph is a rooted unranked tree, without restrictions on the edge directions.
- $G$  is a *DAG* (DAG) if it is a (directed) acyclic graph.

These refine the classes of connected queries considered in [7], by adding the DAG class. We denote by All the class of all graphs. Note that both 2WP and DWT generalize 1WP and are incomparable; PT generalizes both 2WP and DWT; DAG generalizes PT; All generalizes DAG (see Figure 2 of [7]).

**Boolean Provenance.** We use the notion of *Boolean provenance*, or simply *provenance* [20, 3, 30]. In the context of databases, provenance intuitively represents which subsets of the instance satisfy the query: it is used in the intensional approach to probabilistic query evaluation [21]. In this paper, we use provenance to show both upper and lower bounds.

Formally, let  $G = (V_G, E_G, \lambda_G)$  and  $H = (V_H, E_H, \lambda_H)$  be graphs. Seeing  $E_H$  as a set of Boolean variables, a *valuation*  $\nu$  of  $E_H$  is a function  $\nu: E_H \rightarrow \{0, 1\}$  that maps each edge of  $H$  to 0 or 1. Such a valuation  $\nu$  defines a subgraph  $H_\nu$  of  $H$  where we only keep the edges mapped to 1, formally  $H_\nu = (V_H, \{e \in E_H \mid \nu(e) = 1\}, \lambda_H)$ . The *provenance* of  $G$  on  $H$  is then the Boolean function  $\text{Prov}_H^G$  having as variables the edges  $E_H$  of  $H$  and mapping every valuation  $\nu$  of  $E_H$  to 1 (true) or 0 (false) depending on whether  $G \rightsquigarrow H_\nu$  or not. Generalizing this definition, for any integer  $n$ , for any choice of  $a_1, \dots, a_n \in V_G$  and  $b_1, \dots, b_n \in V_H$ , we write  $\text{Prov}_H^G[a_1 := b_1, \dots, a_n := b_n]$  to denote the Boolean function that maps valuations  $\nu$  of  $E_H$  to 1 or 0 depending on whether or not there is a homomorphism  $h: G \rightarrow H_\nu$  which additionally satisfies  $h(a_i) = b_i$  for all  $1 \leq i \leq n$ .

For our lower bounds, we will often seek to represent Boolean formulas as the provenance of queries on graphs:

► **Definition 2.1.** *Given two graphs  $G$  and  $H$ , and a Boolean formula  $\phi$  whose variables  $\{e_1, \dots, e_n\} \subseteq E_H$  are edges of  $H$ , we say that  $\text{Prov}_H^G$  represents  $\phi$  on  $(e_1, \dots, e_n)$  if for every valuation  $\nu: E_H \rightarrow \{0, 1\}$  that maps edges not in  $\{e_1, \dots, e_n\}$  to 1, we have  $\nu \models \phi$  if and only if  $\text{Prov}_H^G(\nu) = 1$ .*

**Circuits and Knowledge Compilation.** We consider representations of Boolean functions in terms of *non-deterministic (ordered) binary decision diagrams*, as well as *decomposable circuits*, which we define below.

A *non-deterministic binary decision diagram* (nBDD) on a set of variables  $V = \{v_1, \dots, v_n\}$  is a rooted DAG  $D$  whose nodes carry a label in  $V \sqcup \{0, 1, \vee\}$  and whose edges can carry an optional label in  $\{0, 1\}$ , subject to the following requirements:

1. there are exactly two leaves (called *sinks*), one labelled by 1 (the *1-sink*), and the other by 0 (the *0-sink*);
2. internal nodes are labelled either by  $\vee$  (called an  *$\vee$ -node*) or by a variable of  $V$  (called a *decision node*); and
3. each decision node has exactly two outgoing edges, labelled 0 and 1; the outgoing edges of  $\vee$ -nodes are unlabelled.

The size  $\|D\|$  of  $D$  is its number of edges. Let  $\nu$  be a valuation of  $V$ , and let  $\pi$  be a path in  $D$  going from the root to one of the sinks. We say that  $\pi$  is *compatible* with  $\nu$  if for every decision node  $n$  of the path, letting  $v \in V$  be the variable labelling  $n$ , then  $\pi$  passes through the outgoing edge of  $n$  labelled with  $\nu(v)$ . In particular, no constraints are imposed at  $\vee$ -nodes; thus, we may have that multiple paths are compatible with a single valuation. The nBDD  $D$  represents a Boolean function, also written  $D$  by abuse of notation, which is defined as follows: for each valuation  $\nu$  of  $V$ , we set  $D(\nu) := 1$  if there exists a path  $\pi$  from the root to the 1-sink of  $D$  that is compatible with  $\nu$ , and set  $D(\nu) := 0$  otherwise. Given an nBDD  $D$  over variables  $V$ , we denote by  $\text{Mods}(D)$  the set of satisfying valuations  $\nu$  of  $D$  such that  $D(\nu) = 1$ , and by  $\text{MC}(D)$  the number  $|\text{Mods}(D)|$  of such valuations. Further, given a rational probability function  $w: V \rightarrow [0, 1]$  on the variables of  $V$ , define  $\text{WMC}(D, w)$  to be the probability that a random valuation  $\nu$  satisfies  $F$ , that is,  $\text{WMC}(D, w) = \sum_{\nu \in \text{Mods}(D)} \prod_{x \in V \text{ s.t. } \nu(x)=1} w(x) \prod_{x \in V \text{ s.t. } \nu(x)=0} (1 - w(x))$ .



In this paper, we primarily focus on a subclass of nBDDs called *non-deterministic ordered binary decision diagrams* (nOBDDs). An nOBDD  $D$  is an nBDD for which there exists a strict total order  $\prec$  on the variables  $V$  such that, for any two decision nodes  $n \neq n'$  such that there is a path from  $n$  to  $n'$ , then, letting  $v$  and  $v'$  be the variables that respectively label  $n$  and  $n'$ , we have  $v \prec v'$ . This implies that, along any path going from the root to a sink, the sequence of variables will be ordered according to  $V$ , with each variable occurring at most once. We use nOBDDs because they admit tractable approximate counting of their satisfying assignments, as we discuss later.

We also show lower bounds on a class of *circuits*, called *decomposable negation normal form* (DNNF) circuits. A *circuit* on a set of variables  $V$  is a directed acyclic graph  $C = (G, W)$ , where  $G$  is a set of *gates*, where  $W \subseteq G \times G$  is a set of edges called *wires*, and where we distinguish an *output gate*  $g_0 \in G$ . The *inputs* of a gate  $g \in G$  are the gates  $g'$  such that there is a wire  $(g', g)$  in  $W$ . The gates can be labelled with variables of  $V$  (called a *variable gate*), or with the Boolean operators  $\vee$ ,  $\wedge$ , and  $\neg$ . We require that gates labelled with variables have no inputs, and that gates labelled with  $\neg$  have exactly one input. A circuit  $C$  defines a Boolean function on  $V$ , also written  $C$  by abuse of notation. Formally, given a valuation  $\nu$  of  $V$ , we define inductively the *evaluation*  $\nu'$  of the gates of  $C$  by setting  $\nu'(g) := \nu(v)$  for a variable-gate  $g$  labelled with variable  $v$ , and setting  $\nu'(g)$  for other gates to be the result of applying the Boolean operators of  $g$  to  $\nu'(g_1), \dots, \nu'(g_n)$  for the inputs  $g_1, \dots, g_n$  of  $g$ . We then define  $C(\nu)$  to be  $\nu'(g_0)$  where  $g_0$  is the output gate of  $C$ .

The circuit is in *negation normal form* if negations are only applied to variables, i.e., for every  $\neg$ -gate, its input is a variable gate. The circuit is *decomposable* if the  $\wedge$ -gates always apply to inputs that depend on disjoint variables: formally, there is no  $\wedge$ -gate  $g$  with two distinct inputs  $g_1$  and  $g_2$ , such that some variable  $v$  labels two variable gates  $g'_1$  and  $g'_2$  with  $g'_1$  having a directed path to  $g_1$  and  $g'_2$  having a directed path to  $g_2$ . A *DNNF* is a circuit which is both decomposable and in negation normal form. Note that we can translate nOBDDs in linear time to DNNFs, more specifically to *structured DNNFs* [4, Proposition 3.8].

**Approximate Weighted Counting for nOBDDs.** Recently, Arenas et al. [9] showed the following result on approximate counting of satisfying assignments of an nOBDD.

► **Theorem 2.2** (Corollary 4.5 of [8]). *Let  $D$  be an nOBDD. Then there exists an FPRAS for computing  $\text{MC}(D)$ .*

For our upper bounds, we need a slight strengthening of this result to apply to *weighted model counting* (WMC) in order to handle probabilities. This can be achieved by translating the approach used in [34, Section 5.1] to the nOBDD setting. We thus show (see Appendix A):

► **Theorem 2.3.** *Let  $D$  be an nOBDD, and  $w : \text{vars}(D) \rightarrow [0, 1]$  be a rational probability function defined on the variables appearing in  $D$ . Then there exists an FPRAS for computing  $\text{WMC}(D, w)$ , running in time polynomial in  $\|D\|$  and  $w$ .*

### 3 Results in the Labelled Setting

We now move on to the presentation of our results. We start with the *labelled* setting of probabilistic graph homomorphism in which the fixed signature  $\sigma$  of the query and instance graph contains more than one label ( $|\sigma| > 1$ ). Our results are summarized in Table 1a.

■ **Table 1** Results on approximation proved in this paper. Key: white ( ) means that the problem lies in P; light grey (■) means that it is #P-hard but admits an FPRAS; dark grey (■) means #P-hardness and non-existence of an FPRAS, assuming  $RP \neq NP$ . All cells without a reference to a corresponding proposition are either implied by one of the other results in this paper, or pertain to exact complexity and were already settled in [7].

(a) Complexity of $\text{PHom}_L(\mathcal{G}, \mathcal{H})$ .						(b) Complexity of $\text{PHom}_U(\mathcal{G}, \mathcal{H})$ .						
$\mathcal{G} \downarrow$	$\mathcal{H} \rightarrow$					$\mathcal{G} \downarrow$	$\mathcal{H} \rightarrow$					
	1WP	2WP	DWT	PT	DAG		All	1WP	2WP	DWT	PT	DAG
1WP					3.1	3.3					4.1	?
2WP			3.7						4.3			
DWT			3.5								4.2	?
PT												

**1WP on DAG.** We start by showing the tractability of approximation for  $\text{PHom}_L(1WP, \text{DAG})$ , which also implies tractability of approximation for  $\text{PHom}_L(1WP, \text{PT})$ , since  $\text{PT} \subseteq \text{DAG}$ .

► **Proposition 3.1.**  *$\text{PHom}_L(1WP, \text{DAG})$  is #P-hard already in data complexity, but it admits an FPRAS.*

For #P-hardness, the result already holds in the unlabelled setting, so it will be shown in Section 4 (see Proposition 4.1). Hence, we focus on the upper bound. We rely on the notion of a *topological ordering* of the edges of a directed acyclic graph  $H = (V, E)$ : it is simply a strict total order  $(E, <)$  with the property that for every consecutive pair of edges  $e_1 = (a_1, a_2)$  and  $e_2 = (a_2, a_3)$ , we have that  $e_1 < e_2$ . Let us fix such an ordering.

**Proof of Proposition 3.1.** We will show that every 1WP query on a DAG instance admits an nOBDD representation of its provenance, which we can compute in combined polynomial time. We can then apply Theorem 2.3, from which the result follows. Let  $G = a_1 \xrightarrow{R_1} \dots \xrightarrow{R_m} a_{m+1}$  be the input path query, and  $H$  the instance graph. We make the following claim:

▷ **Claim 3.2.** For every  $v \in H$ , we can compute in time  $O(\|G\| \times \|H\|)$  an nOBDD representing  $\text{Prov}_L^G[a_1 := v]$  which is ordered by the topological ordering  $<$  fixed above.

*Proof.* Writing  $H = (V, E)$ , we build an nBDD  $D$  consisting of the two sinks and of the following nodes:

- $|V| \times \|G\|$   $\vee$ -nodes written  $n_{u,i}$  for  $u \in V$  and  $1 \leq i \leq m$ ; and
  - $|E| \times \|G\|$  decision nodes written  $d_{e,i}$  for  $e \in E$  and  $1 \leq i \leq m$  which test the edge  $e$ .
- Each  $\vee$ -node  $n_{u,i}$  for  $u \in V$  and  $1 \leq i \leq m$  has outgoing edges to each  $d_{e,i}$  for every edge  $e$  emanating from  $u$  which is labelled  $R_i$ . For each decision node  $d_{e,i}$ , letting  $w$  be the target of edge  $e$ , then  $d_{e,i}$  has an outgoing 0-edge to the 0-sink and an outgoing 1-edge to either  $n_{w,i+1}$  if  $i < m$  or to the 1-sink if  $i = m$ . The root of the nBDD is the node  $n_{v,1}$ .

This construction clearly respects the time bound. To check correctness of the resulting nBDD, it is immediate to observe that, for any path from the root to a sink, the sequence of decision nodes traversed is of the form  $d_{e_1,1}, \dots, d_{e_k,k}$  where the  $e_1, \dots, e_k$  form a path of consecutive edges starting at  $v$  and successively labelled  $R_1, \dots, R_k$ . This implies that the nBDD is in fact an nOBDD ordered by  $<$ . Further, such a path reaches the 1-sink iff  $k = m$  and all decisions are positive, which implies that whenever the nOBDD accepts a subgraph  $H'$  of  $H$  then indeed  $H'$  contains a match of  $G$  mapping  $a_1$  to  $v$ . For the converse direction, we observe that, for any subgraph  $H'$  of  $H$  containing a match of  $G$  mapping  $a_1$  to  $v$ , then,

letting  $e_1, \dots, e_m$  be the successive edges traversed in the match of  $G$ , there is a path from the root of  $D$  to the 1-sink which tests these edges in order. This establishes correctness and concludes the proof of the claim.  $\triangleleft$

Now observe that  $\text{Prov}_{\mathcal{L}H}^G = \text{Prov}_{\mathcal{L}H}^G[a_1 := v_1] \vee \dots \vee \text{Prov}_{\mathcal{L}H}^G[a_1 := v_n]$ , where  $v_1, \dots, v_n$  are precisely the vertices of  $H$ . Thus, it suffices to simply take the disjunction of each nOBDD obtained using the process above across every vertex in  $H$ , which yields in linear time the desired nOBDD. From here we can apply Theorem 2.3, concluding the proof.  $\blacktriangleleft$

**1WP on arbitrary graphs.** We show, however, that tractability of approximation does *not* continue to hold when relaxing the instance class from DAG to arbitrary graphs. This also implies that more expressive classes of query graphs – such as 2WP, DWT, and PT also cannot be tractable to approximate on All instances.

► **Proposition 3.3.**  $\text{PHom}_{\mathcal{L}}(1\text{WP}, \text{All})$  does not admit an FPRAS unless  $\text{RP} = \text{NP}$ .

**Proof.** Our result hinges on the following claim:

▷ **Claim 3.4.** Let  $d > 1$  be a constant. Given a monotone 2-CNF formula  $\phi$  on  $n$  variables where each variable occurs in at most  $d$  clauses, we can build in time  $O(|\phi|)$  a 1WP  $G_\phi$  and All graph  $H_\phi$  containing edges  $(e_1, \dots, e_n)$  such that  $\text{Prov}_{H_\phi}^{G_\phi}$  represents  $\phi$  on  $(e_1, \dots, e_n)$ .

**Proof.** Let  $\phi = \bigwedge_{1 \leq i \leq m} (X_{f_1(i)} \vee X_{f_2(i)})$  be the input CNF instance over the variables  $\{X_1, \dots, X_n\}$ . As we are in the labelled setting, let  $U$  and  $R$  be two distinct labels from the signature. Define the 1WP query graph  $G_\phi$  to be  $\xrightarrow{U} \left( \xrightarrow{R} \xrightarrow{d+2} \xrightarrow{U} \right)^m$ . The instance All graph  $H_\phi$  is defined in the following way:

- For all  $1 \leq i \leq n$ , add an edge  $a_i \xrightarrow{R} b_i$ .
- Add an edge  $c_0 \xrightarrow{U} d_0$  and for each clause  $1 \leq j \leq m$ , an edge  $c_j \xrightarrow{U} d_j$ .
- For each clause  $1 \leq j \leq m$  and variable  $X_i$  occurring in that clause, let  $p$  be the number of this occurrence of  $X_i$  in the formula (i.e., the occurrence of  $X_i$  in the  $j$ -th clause is the  $p$ -th occurrence of  $X_i$ ), with  $1 \leq p \leq d$  by assumption on  $\phi$ . Then add a path of length  $p$  of  $R$ -edges from  $d_{j-1}$  to  $a_i$  and a path of length  $(d+1) - p$  of  $R$ -edges from  $b_i$  to  $c_j$ .

The construction of  $G_\phi$  and  $H_\phi$  is in  $O(|\phi|)$ . Furthermore, notice the following ( $\star$ ). For any  $1 \leq i \leq n$ , the edge  $e = a_i \xrightarrow{R} b_i$  has at most  $d$  incoming  $R$ -paths and  $d$  outgoing  $R$ -paths; the outgoing paths have pairwise distinct *length* (i.e., the number of edges until the next edge is a  $U$ -edge), and likewise for the incoming paths. What is more, each incoming  $R$ -path of length  $p$  corresponds to an outgoing path of length  $(d+1) - p$  and together they connect some  $d_{j-1}$  to some  $c_j$  via the edge  $e$ , where the  $j$ -th clause contains variable  $X_i$ .

Now, define  $(e_1, \dots, e_n)$  to be precisely the edges of the form  $a_i \xrightarrow{R} b_i$  for every  $1 \leq i \leq n$ . Intuitively, the presence or absence of each of these edges corresponds to the valuation of each variable in  $\phi$ . We claim that  $\text{Prov}_{H_\phi}^{G_\phi}$  represents  $\phi$  on  $(e_1, \dots, e_n)$ . It will suffice to show that there is a bijection between the satisfying valuations of  $\phi$ , and the subgraphs of  $H_\phi$  that both (i) contain all the edges not in  $(e_1, \dots, e_n)$ , as these are fixed to 1, and (ii) admit a homomorphism from  $G_\phi$ .

Indeed, consider the bijection defined in the obvious way: keep the edge  $a_i \xrightarrow{R} b_i$  iff  $X_i$  is assigned to true in the valuation. First suppose that some valuation of  $\{X_1, \dots, X_n\}$  satisfies  $\phi$ . Then, for each clause  $1 \leq j \leq m$ , there is a variable in the clause which evaluates to true. We build a match of  $G_\phi$  on the corresponding possible world of  $H_\phi$  by mapping the

## 15:10 Conjunctive Queries on Probabilistic Graphs: The Limits of Approximability

$j$ -th  $U$ -edge to  $c_j \xrightarrow{U} d_j$  for all  $0 \leq j \leq m$ , and mapping the  $R$ -paths for each  $1 \leq j \leq m$  by picking a variable  $X_i$  witnessing that the clause is satisfied and going via the path of length  $1 + (p) + ((d + 1) - p) = d + 2$  that uses the edge  $a_i \xrightarrow{R} b_i$ , which is present by assumption.

Conversely, assume that we have a match of  $G_\phi$  on a possible world of  $H_\phi$ . We show that the corresponding valuation satisfies  $\phi$ . Consider the edge  $c_j \xrightarrow{U} d_j$  to which the first  $U$ -edge is mapped. The  $R$ -path that follows must be mapped to a path from  $d_j$  to some  $a_i$ , and then take the edge  $a_i \xrightarrow{R} b_i$ , whose presence witnesses that the corresponding variable  $X_i$  is true. But importantly, in order for the path to have length precisely  $d + 2$  before reaching the next  $U$ -edge, it must be the case that the length of the path before and after the edge  $a_i \xrightarrow{R} b_i$  sums up to  $d + 1$ . As a result of  $(\star)$ , this is only possible by taking a path that leads to  $c_{j+1} \xrightarrow{U} d_{j+1}$ , and so we know that variable  $X_i$  occurs in the  $j$ -th clause so that clause is satisfied. Repeating the argument shows that all clauses from the  $j$ -th onwards are satisfied, and as we have  $m + 1$   $U$ -edges in the graph  $H_\phi$  and  $m + 1$   $U$ -edges in the graph  $G_\phi$  we know that in fact we must have mapped the first  $U$ -edge to the first  $U$ -edge (i.e.,  $j = 0$ ), and all clauses are satisfied.  $\triangleleft$

By [31, Theorem 2], counting the independent sets of a graph of maximal degree 6 admits an FPRAS only if  $\text{RP} = \text{NP}$ . It is not hard to see that this problem is equivalent to counting satisfying assignments of a monotone 2-CNF formula in which a variable can appear in up to 6 clauses (see, for example, [25, Proposition 1.1]). Thus, we can apply Claim 3.4 above for the class of formulas in which  $d = 6$  to obtain (deterministic) graphs  $G_\phi$  and  $H_\phi$ , and then build a probabilistic graph  $H'_\phi$  identical to  $H_\phi$ , in which the edges  $(e_1, \dots, e_n)$  are assigned probability 0.5 and all other edges probability 1, giving the desired reduction.  $\blacktriangleleft$

**DWT on DWT.** Having classified the cases of one-way path queries (1WP) on all instances classes considered, we turn to more expressive queries. The next two query classes to consider are two-way path queries (2WP) and downward trees queries (DWT). For these query classes, exact computation on 2WP instances is tractable by [7], so the first case to classify is that of DWT instances. Exact computation is intractable in this case by [7], and we show here that, unfortunately, approximation is intractable as well, so that the border for exact tractability coincides with that for approximate tractability. We first focus on DWT queries:

► **Proposition 3.5.**  $\text{PHom}_L(\text{DWT}, \text{DWT})$  does not admit an FPRAS unless  $\text{RP} = \text{NP}$ .

**Proof.** Our result hinges on the following, whose proof adapts [26, Proposition 2.4.3]:

▷ **Claim 3.6.** Given a monotone 2-CNF formula  $\phi$  on  $n$  variables, we can build in time  $O(|\phi| \log |\phi|)$  DWT graphs  $G_\phi$  and  $H_\phi$ , with the latter containing edges  $(e_1, \dots, e_n)$  such that  $\text{Prov}_{H_\phi}^{G_\phi}$  represents  $\phi$  on  $(e_1, \dots, e_n)$ .

**Proof.** Let  $\phi = \bigwedge_{1 \leq i \leq m} (X_{f_1(i)} \vee X_{f_2(i)})$  be the input CNF instance over the variables  $\{X_1, \dots, X_n\}$ . We let  $\bar{L} = \lceil \log_2 m \rceil$  be the number of bits needed to write clause numbers in binary. As we are in the labelled setting, let 0 and 1 be two distinct labels from the signature. Construct the query graph  $G_\phi$  as follows:

- For all  $1 \leq i \leq m$ , add an edge  $z \xrightarrow{0} x_i$ .
- For each  $1 \leq i \leq m$ , letting  $b_1 \cdots b_L$  be the clause number  $i$  written in binary, add a path of  $L$  edges  $x_i \xrightarrow{b_1} y_{i,1} \xrightarrow{b_2} \dots \xrightarrow{b_{L-1}} y_{i,L-1} \xrightarrow{b_L} y_{i,L}$ .

Now, construct the DWT instance  $H_\phi$  as follows:

- For all  $1 \leq i \leq n$ , add the edges  $a \xrightarrow{0} c_i$ .

- For all  $1 \leq i \leq n$  and  $1 \leq j \leq m$  such that  $X_i$  occurs in the  $j$ -th clause of  $\phi$  (i.e.,  $X_i$  is in  $f_1^{-1}(j)$  or  $f_2^{-1}(j)$ ), letting  $b_1 \cdots b_L$  be the clause number  $j$  written in binary, add a path of  $L$  edges  $c_i \xrightarrow{b_1} d_{i,j,1} \xrightarrow{b_2} \cdots \xrightarrow{b_{L-1}} d_{i,j,L-1} \xrightarrow{b_L} d_{i,j,L}$ .

It is clear that  $G_\phi \in \text{DWT}$ ,  $H_\phi \in \text{DWT}$ , and that both graphs can be built in time  $O(|\phi| \log |\phi|)$ . Now, define  $(e_1, \dots, e_n)$  to be the edges of the form  $a \xrightarrow{0} c_i$  for every  $1 \leq i \leq n$ .

We claim that  $\text{Prov}_{H_\phi}^{G_\phi}$  represents  $\phi$  on  $(e_1, \dots, e_n)$ . It suffices to show that there is a bijection between the satisfying valuations  $\nu$  of  $\phi$ , and the subgraphs of  $H_\phi$  that both (i) contain all the edges not in  $(e_1, \dots, e_n)$ , as these are fixed to 1, and (ii) admit a homomorphism from  $G_\phi$ . Indeed, consider the bijection defined in the obvious way: keep the edge  $a \xrightarrow{T} c_i$  iff  $X_i$  is assigned to true in the valuation. First, if there is a homomorphism from  $G_\phi$  to such a subgraph, then the root  $z$  of the query must be mapped to  $a$  (since this is the only element with outgoing paths of length  $L + 1$  as prescribed by the query), and then it is clear that the image of any such homomorphism must take the form of a DWT instance that contains, for each clause number  $1 \leq i \leq m$ , a path of length  $L$  representing this clause number. This witnesses that the valuation  $\nu$  makes a variable true which satisfies clause  $i$ . Hence,  $\nu$  is a satisfying assignment of  $\phi$ . Conversely, for every satisfying assignment  $\nu$ , considering the corresponding subgraph of  $H_\phi$ , we can construct a homomorphism mapping the edges of  $G_\phi$  to the edges of  $H_\phi$ , by mapping the path of every clause to a path connected to a variable that witnesses that this clause is satisfied by  $\nu$ .  $\triangleleft$

The result then follows by an argument analogous to the one in Proposition 3.3.  $\blacktriangleleft$

**2WP on DWT.** We then move to 2WP queries:

► **Proposition 3.7.**  $\text{PHom}_L(2\text{WP}, \text{DWT})$  does not admit an FPRAS unless  $\text{RP} = \text{NP}$ .

This result follows from a general reduction technique from DWT queries on DWT instances to 2WP queries on DWT instances, which allows us to conclude using the result already shown on DWT queries (Proposition 3.5). We note that this technique could also have been used to simplify the proofs of hardness of exact computation in [7] and [2]. We claim:

► **Lemma 3.8.** For any DWT query  $G$ , we can compute in time  $O(\|G\|)$  a 2WP query  $G'$  which is equivalent to  $G$  on DWT instances: for any DWT  $H$ , there is a homomorphism from  $G$  to  $H$  iff there is a homomorphism from  $G'$  to  $H$ .

For lack of space, we give only the construction of  $G'$  here, and defer the full proof of the correctness of this construction to Appendix B.

**Proof.** Let  $G$  be a DWT query. We build  $G'$  following a tree traversal of  $G$ . More precisely, we define the translation inductively as follows. If  $G$  is the trivial query with no edges, then we let the translation of  $G$  be the trivial query with no edges. Otherwise, let  $x$  be the root of  $G$ , let  $x \xrightarrow{R_1} y_1, \dots, x \xrightarrow{R_n} y_n$  be the successive children, and call  $G_1, \dots, G_n$  the DWT subqueries of  $G$  respectively rooted at  $y_1, \dots, y_n$ . We define the translation of  $G$  to be  $\xrightarrow{R_1} G'_1 \xleftarrow{R_1} \cdots \xrightarrow{R_n} G'_n \xleftarrow{R_n}$ , where  $G'_1, \dots, G'_n$  are the respective translations of  $G_1, \dots, G_n$ . This translation is in linear time, and the translated query has twice as many edges as the original query.  $\blacktriangleleft$

Lemma 3.8 allows us to conclude from Proposition 3.5, as it allows us to reduce in linear time (in combined complexity) the evaluation of a DWT query on a DWT probabilistic instance to the evaluation of an equivalent 2WP query on the same instance. This establishes that any approximation algorithm for 2WP queries on DWT instances would give an approximation for DWT queries on DWT instances, which by Proposition 3.5 is conditionally impossible.

## 15:12 Conjunctive Queries on Probabilistic Graphs: The Limits of Approximability

These results complete Table 1, concluding the classification of the complexity of  $\text{PHom}$  in the labelled setting: all cases that were intractable for exact computation are also hard to approximate, with the notable exception of 1WP queries on DAG instances.

### 4 Results in the Unlabelled Setting

We now turn to the *unlabelled* setting of probabilistic graph homomorphism, where the signature  $\sigma$  has only one label ( $|\sigma| = 1$ ). Our results are summarized in Table 1b: we settle all cases except  $\text{PHom}_{\mathcal{L}}(1\text{WP}, \text{All})$  and  $\text{PHom}_{\mathcal{L}}(\text{DWT}, \text{All})$ , for which we do not give an FPRAS or hardness of approximation result. Note that both problems are  $\#\text{P}$ -hard for exact computation [7]. Further, they are in fact equivalent, because DWT queries are equivalent to 1WP queries in the unlabelled setting (stated in [7] and reproved as Proposition 4.2 below).

**1WP on DAG.** We start with 1WP queries, and state the following:

► **Proposition 4.1.**  $\text{PHom}_{\mathcal{L}}(1\text{WP}, \text{DAG})$  is  $\#\text{P}$ -hard already in data complexity, but it admits an FPRAS.

The positive result directly follows from the existence of an FPRAS in the labelled setting, which we have shown in the previous section (Proposition 3.1). By contrast, the  $\#\text{P}$ -hardness does not immediately follow from previous work, as DAG queries were not studied in [7]. We can nevertheless obtain it by inspecting the usual  $\#\text{P}$ -hardness proof of PQE for the CQ  $\exists x y R(x), S(x, y), T(y)$  on TID instances [32]. We give a proof in Appendix C.

**DWT on DAG.** We can easily generalize the above result from 1WP queries to DWT queries, given that they are known to be equivalent in the unlabelled setting:

► **Proposition 4.2** ([7]).  $\text{PHom}_{\mathcal{L}}(\text{DWT}, \text{DAG})$  is  $\#\text{P}$ -hard already in data complexity, but admits an FPRAS.

This is implicit in [7, Proposition 5.5]: we give a self-contained proof in Appendix D.

**2WP on PT.** In contrast to 1WP queries, which are exactly tractable on PT instances and admit an FPRAS on DAG instances, 2WP queries have no FPRAS already on PT instances:

► **Proposition 4.3.**  $\text{PHom}_{\mathcal{L}}(2\text{WP}, \text{PT})$  does not admit an FPRAS unless  $\text{RP} = \text{NP}$ .

**Proof.** It suffices to prove the claim below, which is the analogue to the unlabelled setting of Claim 3.6 after having transformed the query to 2WP via Lemma 3.8:

▷ **Claim 4.4.** Given a monotone 2-CNF formula  $\phi$  on  $n$  variables, we can build in time  $O(|\phi| \log |\phi|)$  an unlabelled 2WP graph  $G_\phi$  and unlabelled PT graph  $H_\phi$ , with the latter containing edges  $(e_1, \dots, e_n)$  such that  $\text{Prov}_{H_\phi}^{G_\phi}$  represents  $\phi$  on  $(e_1, \dots, e_n)$ .

We show this claim via a general-purpose reduction from the labelled setting to the unlabelled setting, which works in fact for All queries on All graphs. This reduction codes labels via specific unlabelled paths; a similar but ad-hoc technique was used to prove [7, Proposition 5.6]:



► **Lemma 4.5.** *For any constant  $k \geq 2$ , given a All query  $G$  and All graph  $H$  on a labelled signature with relation labels  $\{1, \dots, k\}$ , we can construct in linear time an unlabelled All query  $G'$  and All graph  $H'$  such that there is a (labelled) homomorphism from  $G$  to  $H$  iff there is an (unlabelled) homomorphism from  $G'$  to  $H'$ . Further, if  $G$  is a 2WP then  $G'$  is also a 2WP, and if  $H$  is a PT then  $H'$  is also a PT.*

**Proof.** We construct  $G'$  from  $G$  and  $H'$  from  $H$  by replacing every edge by a fixed path that depends on the label of the edge. Specifically, we consider every edge  $x \xrightarrow{i} y$  of the query, where  $x$  is the source,  $t$  is the target, and  $1 \leq i \leq k$  is the label. We code such an edge in  $G'$  by a path defined as follows:  $x \rightarrow^{k+3} \leftarrow \rightarrow^{i+1} \leftarrow^{k+2} y$ , where exponents denote repeated edges and where intermediate vertices are omitted. We code the instance  $H$  to  $H'$  in the same way. This process is clearly linear-time, and it is clear that if  $G$  is a 2WP then  $G'$  is also a 2WP, and that if  $H$  is a PT then  $H'$  is also a PT. Further, to establish correctness of the reduction, one direction of the equivalence is trivial: a homomorphism  $h$  from  $G$  to  $H$  clearly defines a homomorphism from  $G'$  to  $H'$  by mapping the coding in  $G'$  of every edge  $e$  of  $G$  to the coding of the image of  $e$  by  $h$  in  $H'$ .

What is interesting is the converse direction of the equivalence. We establish it via a claim on homomorphic images of the coding of individual edges: for any  $1 \leq i \leq k$ , letting  $e'$  be the coding of an edge  $e = x \xrightarrow{i} y$ , for any homomorphism  $h'$  from  $e'$  to  $H'$ , there must exist an edge  $f = a \xrightarrow{j} b$  in  $H$  such that  $h'$  maps  $x$  to  $a$  and  $y$  to  $b$ . This claim implies the converse direction of the equivalence: if there is a homomorphism  $h'$  from  $G'$  to  $H'$ , then applying the claim to the restrictions of  $h'$  to the coding of each edge of  $G$ , we see that  $h'$  defines a function  $h$  that maps the vertices of  $G$  to vertices of  $H$ , and that  $h$  is a homomorphism. Hence, all that remains is to prove the claim, which we do in the rest of the proof.

Consider an edge  $e = x \xrightarrow{i} y$  as in the claim statement, and let  $e'$  be its coding and  $h'$  the homomorphism mapping  $e'$  to  $H'$ . Observe that, in  $H'$ , the only directed paths of length  $k+3$  are the first  $k+3$  edges of the coding of edges of  $H$ . (This hinges on the fact that the paths of length  $k+3$  defined in the coding of edges of  $H$  are never adjacent in  $H'$  to another edge that goes in the same direction, even across multiple edges, and no matter the directions of edges in  $H$ .) This means that, considering the directed path  $\rightarrow^{k+3}$  at the beginning of  $e'$ , there must be an edge  $f = a \xrightarrow{j} b$  of  $H$ , with coding  $f'$  in  $H'$ , such that the source  $x$  of  $e$  is mapped to the source  $a$  of  $f$ , and the first  $k+3$  edges of  $e'$  are mapped to the first  $k+3$  edges of  $f'$ . What remains to be shown is that  $i = j$  and that  $y$  is mapped to  $b$ .

To this end, we continue studying what can be the image of  $e'$  into  $f'$ . After the directed path  $\rightarrow^{k+3}$ , the next edge  $\leftarrow$  of  $e'$  must have been mapped forward to the next edge  $\leftarrow$  of  $f'$ : indeed, it cannot be mapped backwards on the last edge of the preceding path  $\rightarrow^{k+3}$  because  $k+3 > 1$  and  $i+1 > 1$  so the next edges  $\rightarrow^{i+1}$  would then have no image. Then the next directed path  $\rightarrow^{i+1}$  of  $e'$  is mapped in  $f'$ , necessarily forward because we fail if we map the first edge backwards: this implies that there at least as many edges going in that direction in  $f'$  as there are in  $e'$ , i.e.,  $i \leq j$ . Now, the last path  $\leftarrow^{k+2}$  of  $e'$  cannot be mapped backwards because  $k+2 > i+1$ , so we must map it forwards in  $f'$ : for this to be possible, we must have reached the end of the directed path  $\rightarrow^{j+1}$  in  $f'$ , so that we have  $j = i$ . We are now done reading  $e'$  and  $f'$ , so we have indeed mapped  $y$  to  $b$ . This, along with  $i = j$ , establishes that the claim is true, and concludes the proof. ◀

We can thus prove Claim 4.4, starting from Claim 3.6 and translating it first via Lemma 3.8 and then via Lemma 4.5. Using the same argument as in Proposition 3.3, we conclude the proof of Proposition 4.3. ◀



**5 DNNF Lower Bounds**

In this section, we investigate how to represent the provenance of the query-instance pairs that we consider. More specifically, we study whether there exist polynomially-sized representations in tractable circuit classes of Boolean provenance functions  $\text{Prov}_H^G$ , for  $G \in \mathcal{G}$  and  $H \in \mathcal{H}$  in the graph classes studied in this paper. Certainly, for every graph class  $\mathcal{G}$  and  $\mathcal{H}$ , the (conditional) non-existence of an FPRAS for  $\text{PHom}(\mathcal{G}, \mathcal{H})$  implies that, conditionally, we cannot compute nOBDD representations of provenance in polynomial time combined complexity – as otherwise we could obtain an FPRAS via Theorem 2.3. In fact, beyond nOBDDs, it follows from [9, Theorem 6.3] that, conditionally, we cannot tractably compute provenance representations even in the more general class of *structured DNNFs*. Indeed, as for nOBDDs, fixed edges in the reductions can be handled by conditioning [28, Proposition 4].

However, even in settings where there is conditionally no combined FPRAS, it could be the case that there are polynomial-sized tractable circuits that are difficult to compute, or that we can tractably compute circuits in a more general formalism such as *unstructured* DNNF circuits. The goal of this section is to give a negative answer to these two questions, for all of the non-approximable query-instance class pairs studied in Sections 3 and 4.

Specifically, we show moderately exponential lower bounds on the size of DNNF circuits for infinite families of graphs taken from these classes. Remember that DNNF is arguably the most general knowledge compilation circuit class that still enjoys some tractable properties [16]. Hence, these lower bounds imply that no tractable provenance representation exists in other tractable subclasses of DNNFs, e.g., structured DNNFs [28], or Decision-DNNFs [10]. We also emphasize that, unlike the intractability results of Sections 3 and 4 which assumed  $\text{RP} \neq \text{NP}$ , all of the DNNF lower bounds given here are unconditional.

We first show a *strongly* exponential lower bound for labelled 1WP on All instances:

► **Proposition 5.1.** *There is an infinite family  $G_1, G_2, \dots$  of labelled 1WP queries and an infinite family  $H_1, H_2, \dots$  of labelled All instances such that, for any  $i > 0$ , any DNNF circuit representing the Boolean function  $\text{Prov}_{H_i}^{G_i}$  has size  $2^{\Omega(\|G_i\| + \|H_i\|)}$ .*

**Proof.** By *treewidth* of a monotone 2-CNF formula, we mean the treewidth of the graph on the variables whose edges correspond to clauses in the expected way; and by *degree* we mean the maximal number of clauses in which any variable occurs. Let us consider an infinite family  $\phi_1, \phi_2, \dots$  of monotone 2-CNF formulas of constant degree  $d = 3$  whose treewidth is linear in their size: this exists by [18, Proposition 1, Theorem 5]. We accordingly know by [4, Corollary 8.5] that any DNNF computing  $\phi_i$  must have size  $2^{\Omega(|\phi_i|)}$  for all  $i > 1$ . Using Claim 3.4, we obtain infinite families  $G_1, G_2, \dots$  of 1WP and  $H_1, H_2, \dots$  of All graphs such that  $\text{Prov}_{H_i}^{G_i}$  represents  $\phi_i$  on some choice of edges, and we have  $\|G_i\| + \|H_i\| = O(|\phi_i|)$  for all  $i > 0$  (from the running time bound). Now, any representation of  $\text{Prov}_{H_i}^{G_i}$  as a DNNF can be translated in linear time to a representation of  $\phi_i$  as a DNNF of the same size, simply by renaming the edges  $(e_1, \dots, e_n)$  to the right variables, and replacing all other variables by the constant 1. This means that the lower bound on the size of DNNFs computing  $\phi_i$  also applies to DNNFs representing  $\text{Prov}_{H_i}^{G_i}$ , i.e., they must have size at least  $2^{\Omega(|\phi_i|)}$ , hence  $2^{\Omega(\|G_i\| + \|H_i\|)}$  as we claimed. ◀

We now present lower bounds for the remaining non-approximable query-instance class pairs, which are not exponential but rather *moderately* exponential. This is because our encoding of CNFs into these classes (specifically, Claim 3.6, and its images by Lemma 3.8 and Lemma 4.5) do not give a linear, but rather linearithmic bound. We leave to future work the question of proving strongly exponential lower bounds for these classes, like we did in Proposition 5.1.

► **Proposition 5.2.** *For any  $\epsilon > 0$ , there is an infinite family  $G_1, G_2, \dots$  of labelled DWT queries and an infinite family  $H_1, H_2, \dots$  of labelled DWT instances such that, for any  $i > 0$ , any DNNF circuit representing the Boolean function  $\text{Prov}_{H_i}^{G_i}$  has size at least  $2^{\Omega((\|G_i\| + \|H_i\|)^{1-\epsilon})}$ .*

**Proof.** The proof is identical to that of Proposition 5.1, except that we apply Claim 3.6: for all  $i > 0$ ,  $\|G_i\| + \|H_i\| = O(|\phi_i| \log |\phi_i|)$ . We perform a change of variables: if we write  $y = |\phi_i| \log |\phi_i|$ , then we can show that  $|\phi_i| = e^{W(y)}$ , where  $W$  denotes the Lambert  $W$  function [12]; equivalently  $|\phi_i| = y/W(y)$  as the  $W$  function satisfies  $W(z)e^{W(z)} = z$  for all  $z > 0$ . Thus, the lower bound of  $2^{\Omega(|\phi_i|)}$  on DNNF representations of  $\phi_i$  implies that any DNNF for  $\text{Prov}_{H_j}^{G_j}$  has size at least  $2^{\Omega(\frac{\|G_i\| + \|H_i\|}{W(\|G_i\| + \|H_i\|)})}$ . In particular, as  $W$  grows more slowly than  $n^\epsilon$  for any  $\epsilon > 0$ , this gives a bound of  $2^{\Omega((\|G_i\| + \|H_i\|)^{1-\epsilon})}$  for sufficiently large  $\phi_j$ . ◀

The proof for the following two claims are analogous to that of Proposition 5.2, but using Lemma 3.8 (for the first result) and Claim 4.4 (for the second result):

► **Proposition 5.3.** *For any  $\epsilon > 0$ , there is an infinite family  $G_1, G_2, \dots$  of labelled 2WP queries and an infinite family  $H_1, H_2, \dots$  of labelled DWT instances such that, for any  $i > 0$ , any DNNF circuit representing the Boolean function  $\text{Prov}_{H_i}^{G_i}$  has size at least  $2^{\Omega((\|G_i\| + \|H_i\|)^{1-\epsilon})}$ .*

► **Proposition 5.4.** *For any  $\epsilon > 0$ , there is an infinite family  $G_1, G_2, \dots$  of unlabelled 2WP queries and an infinite family  $H_1, H_2, \dots$  of unlabelled PT instances such that, for any  $i > 0$ , any DNNF circuit representing the Boolean function  $\text{Prov}_{H_i}^{G_i}$  has size at least  $2^{\Omega((\|G_i\| + \|H_i\|)^{1-\epsilon})}$ .*

We finish by remarking that all of the lower bounds above apply to acyclic query classes (i.e., queries of treewidth 1), for which non-probabilistic query evaluation is well-known to be linear in combined complexity [36]. Thus, these results give an interesting example of query classes for which query evaluation is in linear-time combined complexity, but computing even a DNNF representation of query provenance is (moderately) exponential.

## 6 Consequences

In this section, we consider some corollaries and extensions to the results above.

**Optimality of a Previous Result.** Recall from the introduction that, as was shown in [34], PQE for self-join-free conjunctive queries of bounded hypertree width admits a combined FPRAS (in the general setting of probabilistic databases, rather than probabilistic graphs):

► **Theorem 1.3** (Theorem 1 of [34]). *Let  $Q$  be a self-join-free conjunctive query of bounded hypertree width, and  $H$  a tuple-independent database instance. Then there exists a combined FPRAS for computing the probability of  $Q$  on  $H$ , i.e., an FPRAS whose runtime is  $\text{poly}(|Q|, \|H\|, \epsilon^{-1})$ , where  $\epsilon$  is the multiplicative error.*

Can a stronger result be achieved? Our Proposition 4.3 immediately implies the following:

► **Corollary 6.1.** *Assuming  $\text{RP} \neq \text{NP}$ , even on a fixed signature consisting of a single binary relation there is no FPRAS to approximate the probability of an input treewidth-1 CQ on an input treewidth-1 TID instance.*

Hence, tractability no longer holds with self-joins. So, as unbounded hypertree width queries are intractable in combined complexity even for *deterministic* query evaluation, we have:

► **Corollary 6.2.** *The result in Theorem 1.3 is optimal in the following sense: relaxing either the self-join-free or bounded-hypertree-width condition on the query implies the non-existence of a combined FPRAS, unless  $\text{RP} = \text{NP}$ .*

**Network Reliability.** The two-terminal network reliability problem asks the following: given a graph with probabilistic edges and with source and target vertices  $s$  and  $t$ , compute the probability that  $s$  and  $t$  remain connected, assuming independence across edges. Valiant showed that this problem is  $\#\text{P}$ -complete [33, Theorem 1], and Provan and Ball showed that this holds already on directed acyclic graphs [29, Table 1]. Hardness also holds for the related problem of *all-terminal reliability* [29, Table 1], which asks for the probability that the probabilistic graph remains connected as a whole. Given the inherent  $\#\text{P}$ -hardness of these problems, subsequent research has focused on developing tractable approximations.

Although significant progress has been made on FPRASes for all-terminal (un)reliability [19, 23], designing an FPRAS for two-terminal reliability has remained open. This question was even open for the restricted case of directed acyclic graphs; indeed, it was explicitly posed as an open problem by Zenklusen and Laumanns [37]. We now point out that the nOBDD construction of Proposition 3.1 implies an FPRAS for two-terminal reliability on DAGs, again by leveraging the approximate counting result of Arenas et al. [8]:

► **Theorem 6.3.** *There exists an FPRAS for the two-terminal network reliability problem over directed acyclic graphs.*

**Proof.** Given as input an unlabelled probabilistic DAG instance  $H = (V, E)$  and two distinguished source and target vertices  $s$  and  $t \in V$ , construct the labelled DAG instance  $H' = (V, E, \lambda)$  as follows. All vertices and edges are identical to that of  $H$ , but every edge of the form  $(s, x)$  emanating from  $s$  is assigned label  $\lambda((s, x)) = R_s$ , every edge  $(x, t)$  directed towards  $t$  is assigned label  $\lambda((x, t)) = R_t$ , and every other edge  $(x, y)$  is assigned the label  $\lambda((x, y)) = R$ . In the case that  $(s, t) \in E$ , then assign  $\lambda((s, t)) = R'$ .

Now, by the result in Proposition 3.1, we can construct an nOBDD for each of the following  $|E|$  different labelled 1WP queries:  $\frac{R'}{\rightarrow}, \frac{R_s}{\rightarrow} \frac{R_t}{\rightarrow}, \frac{R_s}{\rightarrow} \frac{R}{\rightarrow} \frac{R_t}{\rightarrow}, \dots, \frac{R_s}{\rightarrow} \left( \frac{R}{\rightarrow} \right)^{|E|-2} \frac{R_t}{\rightarrow}$ . All of the nOBDDs have the same ordering (given by a topological ordering of the edges of  $H'$ ), so we may take their disjunction to obtain a (complete) nOBDD  $D$  in linear time, whose accepting paths are in bijection with the  $(s, t)$ -connected valuations of the edges in  $H$ . From here we conclude by applying Theorem 2.3. ◀

We remark that, after submission of this work, Theorem 6.3 was also obtained independently (and with a different approach) in a recent preprint by Feng and Guo [17].

## 7 Conclusions and Future Work

We studied the existence and non-existence of *combined approximation algorithms* for the PQE problem, as well as the existence of polynomially-sized tractable circuit representations of provenance, under the lens of combined complexity.

We see several potential directions for future work. First, it would be interesting to see if the results in Proposition 3.1 and Theorem 6.3 can be extended beyond DAG instances: graph classes of bounded *DAG-width* [11] could be a possible candidate here. We also leave open the problem of filling in the two remaining gaps in Table 1. Namely, we would like to

obtain either an FPRAS or hardness of approximation result for the equivalent problems  $\text{PHom}_\mu(1WP, \text{All})$  and  $\text{PHom}_\mu(\text{DWT}, \text{All})$ . It is also natural to ask whether our results can be lifted from graph signatures to arbitrary relational signatures, or whether they apply in the *unweighted* setting where all edges are required to have the same probability [6, 1, 24]. Another question is whether we can classify the combined complexity of approximate PQE for *disconnected* queries, as was done in [7] in the case of exact computation, for queries that feature disjunction such as UCQs (already in the exact case [7]), or for more general query classes, e.g., with recursion [5].

---

## References

- 1 Antoine Amarilli. Uniform reliability for unbounded homomorphism-closed graph queries. In *ICDT*, 2023. doi:10.4230/LIPIcs.ICDT.2023.14.
- 2 Antoine Amarilli, Pierre Bourhis, Mikaël Monet, and Pierre Senellart. Combined tractability of query evaluation via tree automata and cycluits. In *ICDT*, 2017. doi:10.4230/LIPIcs.ICDT.2017.6.
- 3 Antoine Amarilli, Pierre Bourhis, and Pierre Senellart. Provenance circuits for trees and treelike instances. In *ICALP*, 2015. doi:10.1007/978-3-662-47666-6\_5.
- 4 Antoine Amarilli, Florent Capelli, Mikaël Monet, and Pierre Senellart. Connecting knowledge compilation classes and width parameters. *ToCS*, 2020. doi:10.1007/s00224-019-09930-2.
- 5 Antoine Amarilli and İsmail İlkan Ceylan. The dichotomy of evaluating homomorphism-closed queries on probabilistic graphs. *LMCS*, 2022. doi:10.46298/lmcs-18(1:2)2022.
- 6 Antoine Amarilli and Benny Kimelfeld. Uniform reliability of self-join-free conjunctive queries. *LMCS*, 2022. doi:10.46298/lmcs-18(4:3)2022.
- 7 Antoine Amarilli, Mikaël Monet, and Pierre Senellart. Conjunctive queries on probabilistic graphs: Combined complexity. In *PODS*, 2017. doi:10.1145/3034786.3056121.
- 8 Marcelo Arenas, Luis Alberto Croquevielle, Rajesh Jayaram, and Cristian Riveros. #NFA admits an FPRAS: efficient enumeration, counting, and uniform generation for logspace classes. *J. ACM*, 68(6), 2021. Extended version available as arXiv preprint arXiv:1906.09226 [cs.DS]. doi:10.1145/3477045.
- 9 Marcelo Arenas, Luis Alberto Croquevielle, Rajesh Jayaram, and Cristian Riveros. When is approximate counting for conjunctive queries tractable? In *STOC*. ACM, 2021. Extended version available as arXiv preprint arXiv:2005.10029 [cs.DS]. doi:10.1145/3406325.3451014.
- 10 Paul Beame, Jerry Li, Sudeepa Roy, and Dan Suciu. Exact model counting of query expressions: Limitations of propositional methods. *TODS*, 42(1), 2017. doi:10.1145/2984632.
- 11 Dietmar Berwanger, Anuj Dawar, Paul Hunter, Stephan Kreutzer, and Jan Obdržálek. The DAG-width of directed graphs. *J. Comb. Theory, Ser. B*, 102(4), 2012. doi:10.1016/j.jctb.2012.04.004.
- 12 Robert M. Corless, Gaston H. Gonnet, D. E. G. Hare, David J. Jeffrey, and Donald E. Knuth. On the lambert  $W$  function. *Adv. Comput. Math.*, 5(1), 1996. doi:10.1007/BF02124750.
- 13 Nilesh N. Dalvi and Dan Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, 2004. doi:10.1016/B978-012088469-8.50076-0.
- 14 Nilesh N. Dalvi and Dan Suciu. The dichotomy of probabilistic inference for unions of conjunctive queries. *J. ACM*, 59(6), 2012. doi:10.1145/2395116.2395119.
- 15 Adnan Darwiche. Decomposable negation normal form. *J. ACM*, 48(4), 2001. doi:10.1145/502090.502091.
- 16 Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *J. Artif. Intell. Res.*, 17, 2002. doi:10.1613/jair.989.
- 17 Weiming Feng and Heng Guo. An FPRAS for two terminal reliability in directed acyclic graphs, 2023. doi:10.48550/arXiv.2310.00938.
- 18 Martin Grohe and Dániel Marx. On tree width, bramble size, and expansion. *J. Comb. Theory, Ser. B*, 99(1), 2009. doi:10.1016/j.jctb.2008.06.004.

- 19 Heng Guo and Mark Jerrum. A polynomial-time approximation algorithm for all-terminal network reliability. *SIAM J. Comput.*, 48(3), 2019. doi:10.1137/18M1201846.
- 20 Tomasz Imielinski and Witold Lipski Jr. Incomplete information in relational databases. *J. ACM*, 31(4), 1984. doi:10.1145/1634.1886.
- 21 Abhay Kumar Jha and Dan Suciu. Knowledge compilation meets database theory: Compiling queries to decision diagrams. *ToCS*, 52(3), 2013. doi:10.1007/s00224-012-9392-5.
- 22 Ravi Kannan. Markov chains and polynomial time algorithms. In *FOCS*. IEEE, 1994. doi:10.1109/SFCS.1994.365726.
- 23 David R. Karger. A randomized fully polynomial time approximation scheme for the all-terminal network reliability problem. *SIAM Rev.*, 43(3), 2001. doi:10.1137/S0036144501387141.
- 24 Batya Kenig and Dan Suciu. A dichotomy for the generalized model counting problem for unions of conjunctive queries. In *PODS*, 2021. doi:10.1145/3452021.3458313.
- 25 Jingcheng Liu and Pinyan Lu. FPTAS for counting monotone CNF. In *SODA*. SIAM, 2015. doi:10.1137/1.9781611973730.101.
- 26 Mikaël Monet. *Combined complexity of probabilistic query evaluation. (Complexité combinée d'évaluation de requêtes sur des données probabilistes)*. PhD thesis, University of Paris-Saclay, France, 2018. URL: <https://pastel.archives-ouvertes.fr/tel-01980366>.
- 27 Mikaël Monet. Solving a special case of the intensional vs extensional conjecture in probabilistic databases. In *PODS*. ACM, 2020. doi:10.1145/3375395.3387642.
- 28 Knot Pipatsrisawat and Adnan Darwiche. New compilation languages based on structured decomposability. In *AAAI*. AAAI Press, 2008. URL: <http://www.aaai.org/Library/AAAI/2008/aaai08-082.php>.
- 29 J. Scott Provan and Michael O. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM J. Comput.*, 12(4), 1983. doi:10.1137/0212053.
- 30 Pierre Senellart. Provenance in databases: Principles and applications. In *Reasoning Web*, volume 11810 of *LNCS*. Springer, 2019. doi:10.1007/978-3-030-31423-1\_3.
- 31 Allan Sly. Computational transition at the uniqueness threshold. In *FOCS*. IEEE Computer Society, 2010. doi:10.1109/FOCS.2010.34.
- 32 Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. *Probabilistic Databases. Synthesis Lectures on Data Management*. Morgan & Claypool Publishers, 2011. ISBN: 978-1608456802. doi:10.2200/S00362ED1V01Y201105DTM016.
- 33 Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3), 1979. doi:10.1137/0208032.
- 34 Timothy van Bremen and Kuldeep S. Meel. Probabilistic query evaluation: The combined FPRAS landscape. In *PODS*. ACM, 2023. doi:10.1145/3584372.3588677.
- 35 Moshe Y. Vardi. The complexity of relational query languages (extended abstract). In *STOC*. ACM, 1982. doi:10.1145/800070.802186.
- 36 Mihalis Yannakakis. Algorithms for acyclic database schemes. In *VLDB*. IEEE Computer Society, 1981.
- 37 Rico Zenklusen and Marco Laumanns. High-confidence estimation of small  $s$ - $t$  reliabilities in directed acyclic networks. *Networks*, 57(4), 2011. doi:10.1002/net.20412.

## **A** Proof of Theorem 2.3

► **Theorem 2.3.** *Let  $D$  be an  $n$ OBDD, and  $w : \text{vars}(D) \rightarrow [0, 1]$  be a rational probability function defined on the variables appearing in  $D$ . Then there exists an FPRAS for computing  $\text{WMC}(D, w)$ , running in time polynomial in  $\|D\|$  and  $w$ .*

**Proof.** We may assume without loss of generality that  $D$  contains no variable  $v$  such that  $w(v) = 0$  or  $w(v) = 1$ , since any such variable can be dealt with in constant time by conditioning  $D$  accordingly. We will use the fact that for any positive integer  $n$  and set

of variables  $S = \{x_1, \dots, x_k\}$  such that  $k \geq \lceil \log n \rceil + 1$ , we can construct in time  $O(k)$  a complete OBDD  $C_n(x_1, \dots, x_k)$ , implementing a “comparator” on the variables of  $S$ , that tests if the integer represented by the binary string  $x_1 \dots x_k$  is strictly less than  $n$  (hence,  $C_n(x_1, \dots, x_k)$  has precisely  $n$  satisfying assignments, for any permitted value of  $k$ ).

By [4, Lemma 3.16], we may assume that  $D$  is complete; thus, there is a bijection between the models of the Boolean function captured by  $D$ , and the paths from the root to the 1-sink of  $D$ . Now, complete the following procedure for every variable label  $v_i$  with weight  $w(v_i) = p_i/q_i$  appearing in  $D$ . Set  $k = \lceil \log d \rceil + 1$ , where  $d = \max\{p_i, q_i - p_i\}$ . Send the 1-edge emanating from every node  $r \in D$  labelled with  $v_i$  to the OBDD  $C_{p_i}(x_1, \dots, x_k)$  (where  $x_1, \dots, x_k$  are fresh variables), redirecting edges to the 1-sink of  $C_{p_i}(x_1, \dots, x_k)$  to the original destination of the 1-edge from  $r$ . Do the same for 0-edge from  $r$ , but with the OBDD  $C_{q_i - p_i}(x_1, \dots, x_k)$ . Observe that  $D$  remains a complete nOBDD. Moreover, it is not difficult to see that there are now exactly  $p_i$  paths from the root to the 1-sink of  $D$  that pass through the 1-edge emanating from  $r$ , and  $q_i - p_i$  paths passing through the 0-edge.

After repeating this process for every variable in  $D$ , we may apply Theorem 2.2, before normalizing the result by the product of the weight denominators  $\prod q_i$ . ◀

## B Proof of Lemma 3.8

► **Lemma 3.8.** *For any DWT query  $G$ , we can compute in time  $O(\|G\|)$  a 2WP query  $G'$  which is equivalent to  $G$  on DWT instances: for any DWT  $H$ , there is a homomorphism from  $G$  to  $H$  iff there is a homomorphism from  $G'$  to  $H$ .*

**Proof.** Let  $G$  be a DWT query. We build  $G'$  following a tree traversal of  $G$ . More precisely, we define the translation inductively as follows. If  $G$  is the trivial query with no edges, then we let the translation of  $G$  be the trivial query with no edges. Otherwise, let  $x$  be the root of  $G$ , let  $x \xrightarrow{R_1} y_1, \dots, x \xrightarrow{R_n} y_n$  be the successive children, and call  $G_1, \dots, G_n$  the DWT subqueries of  $G$  respectively rooted at  $y_1, \dots, y_n$ . We define the translation of  $G$  to be  $\xrightarrow{R_1} G'_1 \xleftarrow{R_1} \dots \xrightarrow{R_n} G'_n \xleftarrow{R_n}$ , where  $G'_1, \dots, G'_n$  are the respective translations of  $G_1, \dots, G_n$ . This translation is in linear time, and the translated query has twice as many edges as the original query. Note that we can also inductively define a homomorphism from  $G'$  to  $G$  mapping the first and last elements of  $G'$  to the root of  $G$ : this is immediate in the base case, and in the inductive claim we obtain suitable homomorphisms from each  $G'_i$  to each  $G_i$  by induction and combine them in the expected way.

We claim that, on any DWT instance  $H$ , there is a match of  $G$  mapping the root to  $a$  iff there is a match of  $G'$  mapping both the first variable and last variable of the path to  $a$ . One direction is clear: from the homomorphism presented earlier that maps  $G'$  to  $G$ , we know that any match of  $G$  in  $H$  implies that there is a match of  $G'$  in  $H$  mapping the first and last elements as prescribed. Let us show the converse, and let us actually show by induction on  $G$  a stronger claim: if there is a match of  $G'$  mapping the first variable of  $G'$  to a vertex  $a$ , then the last variable is also mapped to  $a$  and there is a match of  $G$  mapping the root variable to  $a$ . If  $G$  is the vacuous query, then this is immediate: a match of the empty query  $G'$  mapping the first variable to  $a$  must also map the last variable to  $a$  (it is the same variable), and we conclude. Otherwise, let us write  $x$  the root of  $G$  and  $x \xrightarrow{R_1} y_1, \dots, x \xrightarrow{R_n} y_n$  be the children and  $G_1, \dots, G_n$  the subqueries as above. We know that the match of  $G'$  maps the first variable to a vertex  $a$ , and as  $H$  is a DWT instance it maps  $x$  to a child  $a_1$  of  $a$ . Considering  $G_1$  and its translation  $G'_1$ , we notice that we have a match of  $G'_1$  where the first variable is mapped to  $a_1$ . Hence, by induction, the last variable is also mapped to  $a_1$ , and we have a match of  $G_1$  where the root variable is mapped to  $a_1$ . Now, as  $H$  is a DWT



## 15:20 Conjunctive Queries on Probabilistic Graphs: The Limits of Approximability

instance, the next edge  $\xleftarrow{R_1}$  must be mapped to the edge connecting  $a$  and  $a_1$ , so that the next variable in  $G'$  is mapped to  $a$ . Repeating this argument for the successive child edges and child queries in  $G$ , we conclude that the last variable of  $G'$  is mapped to  $a$ , and we obtain matches of  $G_1, \dots, G_n$  that can be combined to a match of  $G$ . ◀

### C Proof of Proposition 4.1

► **Proposition 4.1.**  $\text{PHom}_{\mathcal{L}}(\text{1WP}, \text{DAG})$  is  $\#\text{P}$ -hard already in data complexity, but it admits an FPRAS.

**Proof.** As mentioned in the main text, the positive result directly follows from the existence of an FPRAS in the labelled setting, which was shown in Proposition 3.1. It remains to show  $\#\text{P}$ -hardness here. Consider the following reduction from the  $\#\text{P}$ -hard problem  $\sharp\text{BIS}$ , which asks to count the independent sets of a bipartite undirected graph  $B = (X, Y, E_G)$ . We reduce to  $\text{PHom}_{\mathcal{L}}(G, \text{DAG})$ , where  $G$  is fixed to be the 1WP of length three, i.e.,  $\rightarrow\rightarrow\rightarrow$ .

Construct a probabilistic graph  $H = (\{s, t\} \sqcup X \sqcup Y, E_H)$  and probability labelling  $\pi$ , where  $s$  and  $t$  are fresh vertices, and the edge set  $E_H$  comprises:

- a directed edge  $s \rightarrow a$  for every  $a \in X$ , with probability 0.5;
- a directed edge  $a \rightarrow b$  for every  $(a, b) \in E_G$ , with probability 1 (i.e., the original bipartite graph instance, with every edge directed from  $X$  towards  $Y$ );
- a directed edge  $b \rightarrow s$  for every  $b \in Y$ , with probability 0.5.

We claim that  $\Pr_{\pi}(G \rightsquigarrow H)$  is precisely the number of independent sets of  $B$ , divided by  $|X \sqcup Y|$ . Indeed, just consider the natural bijection between the subgraphs of  $H$  and the independent sets of  $B$ , where for  $a \in X$  we keep the edge  $s \rightarrow a$  iff  $a$  is in the independent set, and for  $b \in Y$  the edge  $b \rightarrow s$  iff  $b$  is in the independent set. ◀

### D Proof of Proposition 4.2

► **Proposition 4.2** ([7]).  $\text{PHom}_{\mathcal{L}}(\text{DWT}, \text{DAG})$  is  $\#\text{P}$ -hard already in data complexity, but admits an FPRAS.

**Proof.** Hardness follows directly from Proposition 4.1, so we show the positive result here. Let  $G$  be a DWT query graph, and  $m$  its height, i.e., the length of the longest directed path it contains. Let  $G'$  be this 1WP of length  $m$ , computable in polynomial time from  $G$ . We claim the following.

▷ **Claim D.1.** For any  $H \in \text{DAG}$ ,  $\text{PHom}_{\mathcal{L}}(G, H) = \text{PHom}_{\mathcal{L}}(G', H)$ .

**Proof.** Certainly, if  $H' \subseteq H$  admits a homomorphism from  $G$ , then it admits one from  $G'$  too since  $G' \subseteq G$ . On the other hand, if  $H'$  admits a homomorphism from  $G'$ , then it also admits one from  $G$ : just map all vertices of distance  $i$  from the root of  $G$  to the image of the  $i$ -th vertex of  $G'$ . ◀

Now the result follows from the FPRAS for  $\text{PHom}_{\mathcal{L}}(G', \text{DAG})$  given by Proposition 3.1. ◀



# Optimally Rewriting Formulas and Database Queries: A Confluence of Term Rewriting, Structural Decomposition, and Complexity

Hubie Chen

Department of Informatics, King's College London, UK

Stefan Mengel

Univ. Artois, CNRS, Centre de Recherche en Informatique de Lens (CRIL), France

---

## Abstract

A central computational task in database theory, finite model theory, and computer science at large is the evaluation of a first-order sentence on a finite structure. In the context of this task, the *width* of a sentence, defined as the maximum number of free variables over all subformulas, has been established as a crucial measure, where minimizing width of a sentence (while retaining logical equivalence) is considered highly desirable. An undecidability result rules out the possibility of an algorithm that, given a first-order sentence, returns a logically equivalent sentence of minimum width; this result motivates the study of width minimization via syntactic rewriting rules, which is this article's focus. For a number of common rewriting rules (which are known to preserve logical equivalence), including rules that allow for the movement of quantifiers, we present an algorithm that, given a positive first-order sentence  $\phi$ , outputs the minimum-width sentence obtainable from  $\phi$  via application of these rules. We thus obtain a complete algorithmic understanding of width minimization up to the studied rules; this result is the first one – of which we are aware – that establishes this type of understanding in such a general setting. Our result builds on the theory of term rewriting and establishes an interface among this theory, query evaluation, and structural decomposition theory.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Logic; Theory of computation  $\rightarrow$  Database query processing and optimization (theory)

**Keywords and phrases** width, query rewriting, structural decomposition, term rewriting

**Digital Object Identifier** 10.4230/LIPIcs.ICDT.2024.16

**Funding** *Hubie Chen*: acknowledges the support of EPSRC grants EP/V039318/1 and EP/X03190X/1.

*Stefan Mengel*: partially supported by the ANR project EQUUS ANR-19-CE48-0019.

**Acknowledgements** The authors thank Carsten Fuhs, Moritz Müller, and Riccardo Treglia for useful feedback and pointers.

## 1 Introduction

The problem of evaluating a first-order sentence on a finite structure is a central computational task in database theory, finite model theory, and indeed computer science at large. In database theory, first-order sentences are viewed as constituting the core of SQL; in parameterized complexity theory, many commonly studied problems can be naturally and directly formulated as cases of this evaluation problem. In the quest to perform this evaluation efficiently, the *width* of a sentence – defined as the maximum number of free variables over all subformulas of the sentence – has become established as a central and crucial measure. A first reason for this is that the natural bottom-up algorithm for sentence evaluation exhibits an exponential dependence on the width [21]; this algorithm runs in polynomial time when restricted to any class of sentences having *bounded width*, meaning that there exists a constant bounding their



© Hubie Chen and Stefan Mengel;  
licensed under Creative Commons License CC-BY 4.0  
27th International Conference on Database Theory (ICDT 2024).

Editors: Graham Cormode and Michael Shekelyan; Article No. 16; pp. 16:1–16:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

width. In addition, there are a number of dichotomy theorems [17, 16, 10, 12] showing that, for particular fragments of first-order logic, a class of sentences admits tractable evaluation if and only if the class has bounded width, up to logical equivalence. These theorems are shown under standard complexity-theoretic assumptions, and concern relational first-order logic, which we focus on in this article.

These results all point to and support the desirability of minimizing the widths of logical sentences. However, an undecidability result [7, Theorem 19] immediately rules out the existence of an algorithm that, given a first-order sentence  $\phi$ , returns a sentence of minimum width among all sentences logically equivalent to  $\phi$ ; indeed, this undecidability result rules out an algorithm that performs as described even on positive first-order sentences. Despite this non-computability result, given the importance of width minimization, it is natural to seek computable methods for reformulating sentences so as to reduce their width. In this article, we show how to perform width minimization, in an optimal fashion, via established *syntactic rewriting rules* known to preserve logical equivalence. The study of such rules is strongly relevant:

- In database theory, such rules are studied and applied extensively to reformulate database queries with the aim of allowing efficient evaluation [1, Chapter 6].
- Well-known tractable cases of conjunctive query evaluation admit logical characterizations via such rules, such as those of bounded treewidth [18] and bounded hypertreewidth [15].
- Such rules play a key role in obtaining the tractability results of some of the mentioned dichotomy theorems [10, 12].

This article focuses on a collection of common and well-known rewriting rules, which includes rules that are well-established in database theory [1, Figures 5.1 and 6.2]. Our main result is the presentation of an algorithm that, given a positive first-order sentence  $\phi$ , outputs a minimum-width sentence obtainable from  $\phi$  via application of the considered rules. We thus obtain a complete algorithmic understanding of width minimization up to the studied rules. Our main result is the first result – of which we are aware – to obtain a comprehensive characterization of width up to syntactic rules in the general setting of positive first-order logic.

In order to obtain our main result, we use the theory of term rewriting; in particular, we make use of basic concepts therein such as *termination* and *confluence*. We view this marriage of *term rewriting* and *query rewriting* as a conceptual contribution of our work. We gently augment the basic theory of term rewriting in two ways. First, we define the notion of a *gauged system*, which is an abstract reduction system along with a *gauge*, a function mapping each element of the system to a number; the gauge represents a quantity that one wants to minimize. In this article our focus is on systems whose elements are first-order formulas, and where the gauge is the width. Second, we define a notion of division of a system by an equivalence relation. This notion allows us to consider equivalence classes w.r.t. a subset of the considered rules, in a sense made precise. In particular, we consider equivalence classes w.r.t. a set of rules that correspond to the computation of tree decompositions (see Theorem 23). This will make precise and transparent the role of tree decomposition computations in our algorithm; this also opens up the possibility of applying procedures that approximate treewidth in a black-box fashion, or decomposition methods that are designed for unbounded-arity query classes, such as those associated with hypertree width [14]. Relatedly, we mention here that for any conjunctive query, the minimum width obtainable by the studied rules is equal to the query’s treewidth, plus one (see Corollary 24); thus, this measure of minimum width, when looked at for positive first-order logic, constitutes a generalization of treewidth.

All together, the presented framework and theory initiate a novel interface among term rewriting, query rewriting, and structural decomposition theory. In our view, the obtention of our main theorem necessitates the development of novel ideas and techniques that draw upon, advance, and fuse together these areas. We believe that further development of this interface has the potential to further unite these areas via asking new questions and prompting new techniques – of service to efficient query evaluation. We wish to emphasize, as a conceptual contribution, the marriage of *term rewriting* and *query rewriting*, which is (to our knowledge) novel, despite an enduring overlap in names!

We wish to add two technical remarks. First, although our results are presented for positive first-order logic, our width minimization procedure can straightforwardly be applied to general first-order sentences by first turning it into negation normal form, i.e., pushing all negations down to the atomic level, and then treating negated atoms as if they were atoms. Second, our main result’s algorithm straightforwardly gives rise to a fixed-parameter tractability result: on any class of sentences that have bounded width when mapped under our algorithm, we obtain fixed-parameter tractability of evaluating the class, with the sentence as parameter, via invoking our algorithm and then performing the natural bottom-up algorithm for sentence evaluation.

## Related work

Some of the previous characterizations and studies of width, for example [10, 12, 8], focused on subfragments of positive first-order logic defined by restricting the permitted quantifiers and connectives; as mentioned, we here consider full positive first-order logic. Width measures for first-order logic have previously been defined [2, 11]. While the measure of [11] makes use of syntactic rewriting rules, and the work [2] studies the relationship of width to syntactic rewriting rules, the present work shows how to optimally minimize formulas up to the set of studied syntactic rewriting rules, a form of result that (to our knowledge) is not entailed by the theory of either of these previous works.<sup>1</sup> In our view, this situation gives our width measure a clear interpretation. This situation also renders some of the previously defined width measures as having, in retrospect, an ad hoc character: as an example, the width measure of [12] has the property that, when a formula has measure  $w$ , it is possible to apply rewriting rules to the formula so that it has width  $w$ , but the optimal width achievable via these rules is not addressed; the considered rule set from [12] is encompassed by the rule set studied here.

## 2 Preliminaries

We assume basic familiarity with the syntax and semantics of relational first-order logic, which is our logic of focus. We assume relational structures to have nonempty universes. In building formulas, we assume that conjunction ( $\wedge$ ) and disjunction ( $\vee$ ) are binary. When  $\psi$  is a first-order formula, we use  $\text{free}(\psi)$  to denote the set of free variables of  $\psi$ . By a *pfo-formula*, we refer to a positive first-order formula, i.e., a first-order formula without any negation. We use PFO to denote the class of all pfo-formulas. The *width* of a first-order formula  $\phi$ , denoted by  $\text{width}(\phi)$ , is defined as the maximum of  $|\text{free}(\psi)|$  over all subformulas  $\psi$  of  $\phi$ . We

<sup>1</sup> We also mention that the class of formulas in this article’s Example 5 has unbounded width under the width measure of [2] (see their Proposition 3.16), but has bounded width under the width notion of the present article. Thus, insofar as the work [2] exploits syntactic rewriting rules, it does not exploit the ones used in Example 5.

will often identify pfo-formulas with their (syntax) trees, that is, trees whose internal nodes are labeled with the operations of first-order logic or labeled with atoms. We also sometimes refer to a node of a syntax tree labeled by an connective  $\oplus \in \{\wedge, \vee\}$  simply as a  $\oplus$ -node. We make the convention that in syntax trees quantifiers and the variables that they bind are in the label of a single node.

## 2.1 Hypergraphs and tree decompositions

A *hypergraph* is a pair  $(V, E)$  where  $V$  is a set called the *vertex set* of the hypergraph, and  $E$  is a subset of the power set of  $V$ , that is, each element of  $E$  is a subset of  $V$ ;  $E$  is called the *edge set* of the hypergraph. Each element of  $V$  is called a *vertex*, and each element of  $E$  is called an *edge*. In this work, we only consider finite hypergraphs, so we tacitly assume that  $V$  is always a finite set. We sometimes specify a hypergraph simply via its edge set  $E$ ; we do this with the understanding that the hypergraph's vertex set is  $\bigcup_{e \in E} e$ . When  $H$  is a hypergraph, we sometimes use  $V(H)$  and  $E(H)$  to denote its vertex set and edge set, respectively. We consider a *graph* to be a hypergraph where every edge has size 2.

A *tree decomposition* of a hypergraph  $H$  is a pair  $(T, \{B_t\}_{t \in V(T)})$  consisting of a tree  $T$  and, for each vertex  $t$  of  $T$ , a *bag*  $B_t$  that is a subset of  $V(H)$ , such that the following conditions hold:

- (vertex coverage) for each vertex  $v \in V(H)$ , there exists a vertex  $t \in V(T)$  such that  $v \in B_t$ ;
- (edge coverage) for each edge  $e \in E(H)$ , there exists a vertex  $t \in V(T)$  such that  $e \subseteq B_t$ ;
- (connectivity) for each vertex  $v \in V(H)$ , the set  $S_v = \{t \in V(T) \mid v \in B_t\}$  induces a connected subtree of  $T$ .

We define the *bagsize* of a tree decomposition  $C = (T, \{B_t\}_{t \in V(T)})$ , denoted by  $\text{bagsize}(C)$ , as  $\max_{t \in V(T)} |B_t|$ , that is, as the maximum size over all bags. The *treewidth* of a hypergraph  $H$  is the minimum, over all tree decompositions  $C$  of  $H$ , of the quantity  $\text{bagsize}(C) - 1$ . A *minimum width tree decomposition* of a hypergraph  $H$  is a tree decomposition  $C$  where  $\text{bagsize}(C) - 1$  is the treewidth of  $H$ .

## 2.2 Term rewriting

### Basics

We here introduce the basic terminology of term rewriting to be used; for the most part, we base our presentation on [3, Chapter 2]. A *system* is a pair  $(D, \rightarrow)$  where  $D$  is a set, and  $\rightarrow$  is a binary relation on  $D$ ; the binary relation  $\rightarrow$  will sometimes be called a *reduction*. Whenever  $\rightarrow$  is a binary relation, we use  $\leftarrow$  to denote its *inverse* (so  $a \rightarrow b$  if and only if  $b \leftarrow a$ ), and we use  $\leftrightarrow$  to denote the union  $\rightarrow \cup \leftarrow$  which is straightforwardly verified to be a symmetric relation. We use  $\xrightarrow{*}$ ,  $\xleftarrow{*}$ , and  $\leftrightarrow^*$  to denote the reflexive-transitive closures of  $\rightarrow$ ,  $\leftarrow$ , and  $\leftrightarrow$ , respectively. Note that  $\leftrightarrow^*$  is an equivalence relation.

We will use the following properties of elements of a system. Let  $(D, \rightarrow)$  be a system, and let  $d, e \in D$ .

- $d$  is *reducible* if there exists  $d' \in D$  such that  $d \rightarrow d'$ .
- $d$  is *in normal form* if it is not reducible.
- $e$  is a *normal form* of  $d$  if  $d \xrightarrow{*} e$  and  $e$  is in normal form.
- $d$  and  $e$  are *joinable*, denoted  $d \downarrow e$ , if there exists an element  $f$  such that  $d \xrightarrow{*} f \xleftarrow{*} e$ .

Relative to a system  $(D, \rightarrow)$ , an element  $d \in D$ , and a binary relation  $\rightarrow'$ , we say that  $\rightarrow'$  is *applicable to  $d$*  or *can be applied to  $d$*  if there exists an element  $e \in D$  such that  $d \rightarrow' e$ .

We next define properties of a system  $Y = (D, \rightarrow)$ ; in what follows,  $d, e, e', d_i$ , etc. denote elements of  $D$ .

- $Y$  is *confluent* if  $e \xleftarrow{*} d \xrightarrow{*} e'$  implies  $e \downarrow e'$ .
- $Y$  is *locally confluent* if  $e \leftarrow d \rightarrow e'$  implies  $e \downarrow e'$ .
- $Y$  is *terminating* if there is no infinite descending chain  $d_0 \rightarrow d_1 \rightarrow \dots$  of elements in  $D$ .
- $Y$  is *convergent* if it is confluent and terminating.

The property of being terminating immediately implies that of being *normalizing*, meaning that each element has a normal form.

We will use the following properties of convergent systems.

► **Proposition 1** (see [3], Lemma 2.1.8 and Theorem 2.1.9). *Suppose that  $(D, \rightarrow)$  is a convergent system. Then, each element  $d \in D$  has a unique normal form. Moreover, for all elements  $d, e \in D$ , it holds that  $d \xleftrightarrow{*} e$  if and only if they have the same normal form.*

Whenever an element  $d \in D$  has a unique normal form, we denote this unique normal form by  $d \downarrow$ . The last part of Proposition 1 can then be formulated as follows: for every convergent system  $(D, \rightarrow)$  we have for all elements  $d, e \in D$  that  $d \xleftrightarrow{*} e$  if and only if  $d \downarrow = e \downarrow$ .

The following lemma, often called *Newman's Lemma*, was first shown in [19] and will be used to establish convergence. See also [3, Lemma 2.7.2] for a proof.

► **Lemma 2** ([19]). *If a system is locally confluent and terminating, then it is confluent, and hence convergent.*

### Gauged systems

We next extend the usual setting of term rewriting by considering systems having a *gauge*, which intuitively is a measure that one desires to minimize by rewriting. To this end, define a *gauged system* to be a triple  $(D, \rightarrow, g)$  where  $(D, \rightarrow)$  is a system, and  $g : D \rightarrow \mathbb{R}$  is a mapping called a *gauge*. A gauged system is *monotone* if for any pair of elements  $d, e \in D$  where  $d \rightarrow e$ , it holds that  $g(d) \geq g(e)$ , that is, applying the reduction cannot increase the gauge. We apply the terminology of Section 2.2 to gauged systems; for example, we say that a gauged system  $(D, \rightarrow, g)$  is *convergent* if the system  $(D, \rightarrow)$  is convergent.

► **Proposition 3.** *Let  $(D, \rightarrow, g)$  be a gauged system that is monotone and convergent. Then, for any elements  $d, e \in D$ , it holds that  $d \xleftrightarrow{*} e$  implies  $g(d \downarrow) \leq g(e)$ . That is, for any element  $d$ , its normal form  $d \downarrow$  has the minimum gauge among all elements  $e$  with  $d \xleftrightarrow{*} e$ .*

**Proof.** First note that, by a simple induction using the monotonicity of  $g$ , we have for all elements  $e', e'' \in D$  that  $e' \xrightarrow{*} e''$  implies  $g(e') \geq g(e'')$ . Next, since  $(D, \rightarrow)$  is convergent, we have by Proposition 1 that  $d \downarrow = e \downarrow$ . But then  $g(e) \geq g(e \downarrow) = g(d \downarrow)$ . ◀

### Systems and division

We here define a notion of dividing systems by equivalence relations. Let  $D$  be a set and let  $\equiv$  be an equivalence relation on  $D$ . Let  $D/\equiv$  denote the set containing the  $\equiv$ -equivalence classes. We define  $(D, \rightarrow)/\equiv$  as the system  $(D/\equiv, \rightarrow)$ , where we extend the definition of  $\rightarrow$  to  $D/\equiv$  by positing that for  $\equiv$ -equivalence classes  $C, C'$ , it holds that  $C \rightarrow C'$  if and only if there exist  $d \in C$  and  $d' \in C'$  such that  $d \rightarrow d'$ .

Suppose that  $G = (D, \rightarrow, g)$  is a gauged system. We extend the definition of  $g$  so that, for each set  $S \subseteq D$ , we have  $g(S) = \inf\{g(d) \mid d \in S\}$ . We define  $G/\equiv$  as the system  $(D/\equiv, \rightarrow, g)$ .

### 3 Rewriting rules

We here define well-known transformation rules on first-order formulas, which are all known to preserve logical equivalence. We define these rules with the understanding that they can always be applied to subformulas, so that when  $\psi_1$  is a subformula of  $\phi_1$  and  $\psi_1 \rightarrow \psi_2$ , we have  $\phi_1 \rightarrow \phi_2$ , where  $\phi_2$  is obtained from  $\phi_1$  by replacing the subformula  $\psi_1$  with  $\psi_2$ . In the following  $F_1, F_2$ , etc. denote formulas. It will be convenient to formalize these rules as rewriting rules in the sense defined before.

The rule  $\rightarrow_A$  is *associativity* for  $\oplus \in \{\wedge, \vee\}$ :

$$(F_1 \oplus (F_2 \oplus F_3)) \rightarrow_A ((F_1 \oplus F_2) \oplus F_3), \quad ((F_1 \oplus F_2) \oplus F_3) \rightarrow_A (F_1 \oplus (F_2 \oplus F_3)).$$

The rule  $\rightarrow_C$  is *commutativity* for  $\oplus \in \{\wedge, \vee\}$ :

$$(F_1 \oplus F_2) \rightarrow_C (F_2 \oplus F_1).$$

The *pushdown* rule  $\rightarrow_{P\downarrow}$  applies when  $F_2$  is a formula in which  $x$  is not free:

$$\exists x(F_1 \wedge F_2) \rightarrow_{P\downarrow} (\exists x F_1) \wedge F_2, \quad \forall x(F_1 \vee F_2) \rightarrow_{P\downarrow} (\forall x F_1) \vee F_2.$$

The *pushup* rule  $\rightarrow_{P\uparrow}$  is defined as the inverse of  $\rightarrow_{P\downarrow}$ .

The *reordering* rule  $\rightarrow_O$  is defined when  $Q \in \{\forall, \exists\}$  is a quantifier:  $QxQyF \rightarrow_O QyQxF$ . The *renaming* rule  $\rightarrow_N$  allows  $QxF \rightarrow_N QyF'$  when  $y \notin \text{free}(F)$  and  $F'$  is derived from  $F$  by replacing each free occurrence of  $x$  with  $y$ .

Collectively, we call the above rules the *tree decomposition rules*, since, as we will see in Section 8, they allow, in a way that we will make precise, optimizing the treewidth of certain subformulas.

The *splitdown* rule distributes quantification over a corresponding connective:

$$\exists x(F_1 \vee F_2) \rightarrow_{S\downarrow} (\exists x F_1) \vee (\exists x F_2), \quad \forall x(F_1 \wedge F_2) \rightarrow_{S\downarrow} (\forall x F_1) \wedge (\forall x F_2).$$

The *splitup* rule  $\rightarrow_{S\uparrow}$  is defined as the inverse of  $\rightarrow_{S\downarrow}$ .

The *removal* rule  $\rightarrow_M$  allows removal of a quantification when no variables are bound to the quantification: when  $Q \in \{\forall, \exists\}$ , we have  $QxF \rightarrow_M F$  when  $F$  contains no free occurrences of the variable  $x$ , that is, when  $x \notin \text{free}(F)$ .

We tacitly use the straightforwardly verified fact that when the given rules are applied to any pfo-formula, the formula's set of free variables is preserved. We denote these rules via the given subscripts. We use  $\mathcal{T}$  to denote the set of all tree decomposition rules, so  $\mathcal{T} = \{A, C, P\downarrow, P\uparrow, O, N\}$ . We use  $\mathcal{A}$  to denote the set of all presented rules, so  $\mathcal{A} = \mathcal{T} \cup \{S\downarrow, S\uparrow, M\}$ .

When we have a set  $\mathcal{R}$  of subscripts representing these rules, we use  $\rightarrow_{\mathcal{R}}$  to denote the union  $\bigcup_{R \in \mathcal{R}} \rightarrow_R$ . For example,  $\rightarrow_{\{A, C, O\}}$  denotes  $\rightarrow_A \cup \rightarrow_C \cup \rightarrow_O$ . In this context, we sometimes denote a set by the string concatenation of its elements, for example, we write  $\rightarrow_{ACO}$  in place of  $\rightarrow_{\{A, C, O\}}$ . We apply these conventions to  $\leftarrow, \leftrightarrow, \overset{*}{\rightarrow}, \overset{*}{\leftarrow}, \overset{*}{\leftrightarrow}$ , and so forth. When  $\mathcal{R}$  is a set of rule subscripts and  $\phi$  is a formula, we use  $[\phi]_{\mathcal{R}}$  to denote the  $\overset{*}{\leftrightarrow}_{\mathcal{R}}$ -equivalence class of  $\phi$ . So, for example,  $[\phi]_{\mathcal{T}}$  denotes the  $\overset{*}{\leftrightarrow}_{\mathcal{T}}$ -equivalence class of  $\phi$ , and  $[\phi]_{ACO}$  denotes the  $\overset{*}{\leftrightarrow}_{ACO}$ -equivalence class of  $\phi$ .

► **Example 4.** Let  $\phi = \exists x \exists y \exists t (R(x, t) \wedge S(t, y))$ ; this sentence has width 3. We have

$$\phi \overset{*}{\rightarrow}_O \exists t \exists x \exists y (R(x, t) \wedge S(t, y)) \overset{*}{\rightarrow}_{P\downarrow} \exists t ((\exists x R(x, t)) \wedge (\exists y S(t, y))).$$

The last formula has width 2; the rules we used were in  $\mathcal{T}$ . It can, however, be verified that, using the rules in  $\mathcal{T} \setminus \{O\}$ , this formula  $\phi$  cannot be rewritten into a width 2 formula: using these rules, any rewriting will be derivable from  $\phi$  via commutativity and renaming.

► **Example 5.** Consider the formulas  $(\phi_n)_{n \geq 1}$  defined by  $\phi_n = \exists x_1 \dots \exists x_n \forall y (\bigwedge_{i=1}^n E_i(x_i, y))$ ; the formula  $\phi_n$  has width  $n + 1$ . (Although formally we consider conjunction as a binary connective, we allow higher-arity conjunctions due to the presence of the associativity rule.) We have

$$\phi_n \xrightarrow{*}_{S \downarrow, A} \exists x_1 \dots \exists x_n \left( \bigwedge_{i=1}^n \forall y E_i(x_i, y) \right) \xrightarrow{*}_{P \downarrow, A} \bigwedge_{i=1}^n (\exists x_i \forall y E_i(x_i, y)).$$

The last formula has width 2, and hence the rules we consider suffice to convert each formula  $\phi_n$  into a width 2 formula.

### Justification of the rule choice

Before stating our main result in the next section, let us take some time to discuss why we have chosen the above rules for study, in this paper. First, as suggested in the article, these rules appear in a number of textbooks and well-known sources. In addition to being in the standard database book [1], one can find some of the crucial ones in [20, page 99], and many of the crucial ones also appear in the Wikipedia entry on first-order logic.<sup>2</sup> We can remark that these rules are generally used when showing that first-order formulas can be rewritten into prenex normal form.

Indeed, apart from distributivity (discussed in the conclusion), we are not aware of any other syntactic rewriting rules (apart from combinations of the rules that we study); in particular, we did not find any others in any of the standard sources that we looked at.

As alluded to in the introduction, the study of many of these rules is strongly established in the research literature. They are studied in articles including [2, 7, 12, 10, 11, 13, 15, 18]. As mentioned, subsets of the rule set are used crucially, in the literature, to obtain the positive results of dichotomy theorems. In particular, there are precise contexts where a subset of the rule set is sufficient to give a tractability result (for example, in [12]). The tractability result of the present article strengthens these previous tractability results since it optimally minimizes width with respect to the considered rules, and thus provides a wider and deeper perspective on these previous tractability results.

Let us emphasize that a subset of the considered rules corresponds to treewidth computation in a very precise sense (see Theorem 23 and Corollary 24). The correspondence between rewriting rules and computation of tree decompositions was also studied, for example, in [18, 13, 15].

## 4 Main theorem and roadmap

In this section, we first state our main theorem, then we give the intermediate results that we will show in the remainder of the paper to derive it; at the end of the section, we prove the main theorem using the intermediate results. Our main theorem essentially says the following: there is an algorithm that, given a pfo-formula  $\phi$ , computes a formula  $\phi'$  that is derived from  $\phi$  by applying the defined rules  $\mathcal{A}$  and that has the minimum width over formulas derivable by these rules. Moreover, up to computation of minimum width tree decompositions, the algorithm computing  $\phi$  is efficient.

<sup>2</sup> [https://en.wikipedia.org/wiki/First-order\\_logic#Provable\\_identities](https://en.wikipedia.org/wiki/First-order_logic#Provable_identities)



► **Theorem 6** (Main theorem). *There exists an algorithm  $A$  that, given a pfo-formula  $\phi$ , computes a minimum width element of  $[\phi]_{\mathcal{A}}$ . Moreover, with oracle access to an algorithm for computing minimum width tree decompositions, the algorithm  $A$  can be implemented in polynomial time.*

Here, we say that an algorithm *computes minimum width tree decompositions* if, when given a hypergraph  $H$ , it outputs a minimum width tree decomposition of  $H$ .

In order to establish our main theorem, we study and make use of the systems

$$Y = (\text{PFO}, \rightarrow_{\{P\downarrow, S\downarrow, M\}}) / \overset{*}{\leftrightarrow}_{ACO}, \quad Y' = (\text{PFO}, \rightarrow_{\{S\downarrow, M\}}) / \overset{*}{\leftrightarrow}_{\mathcal{T}}.$$

We also make use of the gauged system

$$G' = (\text{PFO}, \rightarrow_{\{S\downarrow, M\}}, \text{width}) / \overset{*}{\leftrightarrow}_{\mathcal{T}},$$

where, as defined before, **width** is the function that maps a formula to its width. We show that for the system  $Y$ , normal forms can be computed in polynomial time, and moreover, that a normal form of  $Y$  directly yields a normal form of  $Y'$ .

► **Theorem 7.** *The system  $Y$  is terminating; moreover, there exists a polynomial-time algorithm that, given a pfo-formula  $\phi$ , returns a pfo-formula  $\phi^+$  such that  $[\phi^+]_{ACO}$  is a normal form of  $[\phi]_{ACO}$  in the system  $Y$ .*

► **Theorem 8.** *Suppose that  $\phi$  is a pfo-formula where  $[\phi]_{ACO}$  is a normal form of the system  $Y$ ; then,  $[\phi]_{\mathcal{T}}$  is a normal form of the system  $Y'$ .*

We then prove that the system  $Y'$  is convergent, and that its corresponding gauged system  $G'$  is monotone. Together, these results allow us to leverage Proposition 3 and allow us to compute minimum-width equivalence classes in  $G'$ .

► **Theorem 9.** *The system  $Y'$  is convergent.*

► **Theorem 10.** *The gauged system  $G'$  is monotone.*

The remaining piece needed is to show that minimization can be performed within a  $\overset{*}{\leftrightarrow}_{\mathcal{T}}$ -equivalence class, which is what the following theorem supplies.

► **Theorem 11.** *There exists an algorithm that, given a pfo-formula  $\theta$ , outputs a pfo-formula  $\theta^+$  having minimum width among all formulas in  $[\theta]_{\mathcal{T}}$ . With oracle access to an algorithm for computing minimum width tree decompositions, this algorithm can be implemented in polynomial time.*

**Proof of the main theorem – Theorem 6.** Given a pfo-formula  $\phi$ , the algorithm first applies the algorithm of Theorem 7 to obtain a pfo-formula  $\theta$  where  $[\theta]_{ACO}$  is a normal form of  $[\phi]_{ACO}$  in  $Y$ , and then applies the algorithm of Theorem 11 to obtain a pfo-formula  $\theta^+$  having minimum width among the formulas in  $[\theta]_{\mathcal{T}}$ ;  $\theta^+$  is the output of the algorithm.

We justify this algorithm's correctness as follows. As  $[\theta]_{ACO}$  is a normal form of  $[\phi]_{ACO}$  in  $Y$ , we have  $[\phi]_{ACO} \overset{*}{\rightarrow}_{\{P\downarrow, S\downarrow, M\}} [\theta]_{ACO}$ , and thus  $[\phi]_{\mathcal{T}} \overset{*}{\rightarrow}_{\{S\downarrow, M\}} [\theta]_{\mathcal{T}}$ , because all applications of  $P\downarrow$  can be simulated by choosing a representative in the equivalence class w.r.t.  $\mathcal{T}$ . It follows from Theorem 8 that  $[\theta]_{\mathcal{T}}$  is a normal form of  $[\phi]_{\mathcal{T}}$  in  $Y'$ . From Theorems 9 and 10, we have that the gauged system  $G'$  is convergent and monotone, so by Proposition 3, we have that, in the gauged system  $G'$ , the element  $[\theta]_{\mathcal{T}}$  has the minimum gauge among all elements that are  $\overset{*}{\leftrightarrow}_{\{S\downarrow, M\}}$ -related to  $[\phi]_{\mathcal{T}}$ . Thus, a minimum width element in  $[\theta]_{\mathcal{T}}$  is a minimum width element in  $[\phi]_{\mathcal{T} \cup \{S\downarrow, S\uparrow, M\}}$ . ◀

## 5 Formulas

In this section, we define a few types of formulas to be used in our development, and show some basic properties thereof.

### 5.1 Holey formulas

A *holey formula* is intuitively defined similarly to a formula, but in lieu of atoms, there are placeholders where further formulas can be attached; these placeholders are represented by natural numbers. Formally, a *holey pfo-formula* is a formula built as follows: each natural number  $i \geq 1$  is a holey pfo-formula, and is referred to as a *hole*; when  $\phi$  and  $\phi'$  are holey pfo-formulas, so are  $\phi \wedge \phi'$  and  $\phi \vee \phi'$ ; and, when  $\phi$  is a holey pfo-formula and  $x$  is a variable, so are  $\exists x\phi$  and  $\forall x\phi$ . We require that for each holey pfo-formula  $\phi$ , no natural number occurs more than once. We say that a holey pfo-formula is *atomic* if it is equal to a natural number (equivalently, if it contains no connectives nor quantifiers). A *holey  $\{\exists, \wedge\}$ -formula* is defined as a holey pfo-formula where, apart from atoms, only the formation rules involving existential quantification and conjunction are permitted. A *holey  $\{\forall, \vee\}$ -formula* is defined dually.

When  $\phi$  is a holey formula with holes among  $1, \dots, k$  and we have that  $\psi_1, \dots, \psi_k$  are formulas, we use  $\phi[\psi_1, \dots, \psi_k]$  to denote the formula obtained from  $\phi$  by substituting, for each  $i = 1, \dots, k$ , the formula  $\psi_i$  in place of  $i$ .

► **Example 12.** Consider the holey pfo-formula  $\phi = 2 \wedge 1$ , and the formulas  $\psi_1 = S(x) \vee S(y)$ ,  $\psi_2 = R(x, z)$ . We have  $\phi[\psi_1, \psi_2] = R(x, z) \wedge (S(x) \vee S(y))$ .

In the sequel, we will speak of applying rewriting rules (generally excluding the rule  $N$ ) to holey formulas. In order to speak of the applicability of rules such as the pushdown and pushup rules, we need to associate a set of free variables to each subformula of a holey formula. We define an *association* for a holey formula  $\phi$  to be a partial mapping  $a$  defined on the natural numbers that is defined on each hole in  $\phi$ , and where, for each number  $i$  on which  $a$  is defined, it holds that  $a(i)$  is a set of variables. With an association  $a$  for a holey formula  $\phi$ , we can naturally define a set of free variables on each subformula of  $\phi$ , by considering  $\text{free}(i) = a(i)$  for each  $i$  on which  $a$  is defined, and then using the usual inductive definition of  $\text{free}(\cdot)$ .

### 5.2 Standardized formulas

Define a *standardized* formula to be a formula  $\phi$  where, for each occurrence  $Qx$  of quantification, the following hold: (1)  $x$  is not quantified elsewhere, that is, for any other occurrence  $Q'x'$  of quantification,  $x \neq x'$  holds; (2)  $x \notin \text{free}(\phi)$ . Intuitively, a standardized formula is one where there is no name clash between a quantified variable  $x$  and other variable occurrences in the formula. It is straightforward to verify that a subformula of a standardized formula is also standardized.

It is straightforward to verify that every first-order formula can be standardized by repeatedly applying the renaming rule. We will use the following formalization of this observation which in effect asserts that in terms of applying the studied rules to reduce width, one may always assume that variable renaming that leads to a standardized formula is always performed upfront.

► **Proposition 13.** *Let  $\phi$  be a pfo-formula or a holey pfo-formula along with an association  $a$ . Suppose  $\phi = \phi_0$  and  $\phi_0 \rightarrow_{A_1} \phi_1 \cdots \rightarrow_{A_t} \phi_t$  where each  $A_i$  is in  $\mathcal{T} \cup \{S \downarrow, M\}$ . Let  $A'_1, \dots, A'_s$  be the sequence obtained from  $A_1, \dots, A_t$  by removing instances of  $N$ . Then, there exists a*

standardized formula  $\phi'_0$  with the following property: there exist formulas  $\phi'_1, \dots, \phi'_s$  where  $\phi \xrightarrow{*}_N \phi'_0 \rightarrow_{A'_1} \phi'_1 \cdots \rightarrow_{A'_s} \phi'_s \xrightarrow{*}_N \phi_t$ . Indeed, for any standardized formula  $\phi'_0$  with  $\phi \xrightarrow{*}_N \phi'_0$ , the given property holds.

### 5.3 Organized formulas

We next define and study *organized formulas*, which, intuitively speaking, are pfo-formulas that are stratified into regions, based on the quantifiers and connectives. The first main property we show is that each non-atomic pfo-formula can be viewed as an organized formula.

We define  $\{\exists, \wedge\}$ -organized formulas and  $\{\forall, \vee\}$ -organized formulas by mutual induction, as follows.

- When  $\phi$  is a non-atomic holey  $\{\exists, \wedge\}$ -formula and each of  $\psi_1, \dots, \psi_k$  is an atom or a  $\{\forall, \vee\}$ -organized formula, then  $\phi[\psi_1, \dots, \psi_k]$  is an  $\{\exists, \wedge\}$ -organized formula.
- When  $\phi$  is a non-atomic holey  $\{\forall, \vee\}$ -formula and each of  $\psi_1, \dots, \psi_k$  is an atom or an  $\{\exists, \wedge\}$ -organized formula, then  $\phi[\psi_1, \dots, \psi_k]$  is a  $\{\forall, \vee\}$ -organized formula.

An *organized formula* is defined to be a formula that is either an  $\{\exists, \wedge\}$ -organized formula or a  $\{\forall, \vee\}$ -organized formula.

We call the formula  $\phi$  in the above definition a *region* of the organized formula. Thus, every organized formula  $\theta$  can be decomposed into atoms and different regions which are maximal holey  $\{\exists, \wedge\}$ - and  $\{\forall, \vee\}$ -subformulas of  $\theta$ . Define the *top operation* of a pfo-formula to be the label of the root of the formula when interpreting it as a tree.

► **Proposition 14.** *Each non-atomic pfo-formula  $\theta$  is an organized formula. More precisely, each non-atomic pfo-formula  $\theta$  whose top operation is  $\exists$  or  $\wedge$  is an  $\{\exists, \wedge\}$ -organized formula. Each non-atomic pfo-formula  $\theta$  whose top operation is  $\forall$  or  $\vee$  is an  $\{\forall, \vee\}$ -organized formula.*

When the rules in  $\mathcal{T} \setminus \{N\}$  are applied to organized formulas, they act on regions independently, in the following formal sense.

► **Proposition 15.** *Suppose that  $\Phi = \phi[\psi_1, \dots, \psi_k]$  is an organized formula. Then, each formula in  $[\Phi]_{\mathcal{T} \setminus \{N\}}$  has the form  $\phi'[\psi'_1, \dots, \psi'_k]$ , where  $\phi' \in [\phi]_{\mathcal{T} \setminus \{N\}}$  and  $\psi'_1 \in [\psi_1]_{\mathcal{T} \setminus \{N\}}, \dots, \psi'_k \in [\psi_k]_{\mathcal{T} \setminus \{N\}}$ . Here, we understand the  $\mathcal{T} \setminus \{N\}$ -rules to be applied to  $\phi$  and holey pfo-formulas under the association  $i \mapsto \text{free}(\psi_i)$  defined on each  $i = 1, \dots, k$ .*

## 6 Rule applicability

In the remainder of the paper, it will be crucial to have an understanding for when the rules of the systems  $Y$  and  $Y'$  can be applied. To this end, in this section, we study the structure of formulas that allow their application. This will in particular also lead to the normal-form result of Theorem 8.

We start with the applicability of the removal rule  $\rightarrow_M$ . Remember that we say that a rule  $\rightarrow$  can be applied to a set  $\Phi$  of formulas if and only if there is a formula  $\phi \in \Phi$  on which  $\rightarrow$  is applicable.

► **Lemma 16.** *Let  $\theta$  be a pfo-formula. Then the following statements are equivalent:*

- $\rightarrow_M$  can be applied to  $\theta$ .
- $\rightarrow_M$  can be applied to  $[\theta]_{ACO}$ .
- $\rightarrow_M$  can be applied to  $[\theta]_{\mathcal{T}}$ .

**Proof sketch.** The proof is based on the simple observation that a quantifier does not bind a variable in  $\theta$  if and only if it does not bind a variable in any representative in  $[\theta]_{ACO}$  and  $[\theta]_{\mathcal{T}}$ . ◀

It will be useful to consider (sets of) formulas in which the pushdown operation has been applied exhaustively, in particular, to understand normal forms of the system  $Y$ . To this end, we introduce the following definitions. We say that a pfo-formula  $\phi$  is *pushed-down* if  $\rightarrow_{P\downarrow}$  cannot be applied to  $\phi$ ; we say that a set  $\Phi$  of pfo-formulas is *pushed-down* if each formula  $\phi \in \Phi$  is pushed-down.

We say that a pfo-formula is a *k-fold conjunction* if, up to associativity, it can be written in the form  $\psi_1 \wedge \dots \wedge \psi_k$ . We can formalize this as follows. For each pfo-formula  $\theta$ , define the multiset  $\text{conjuncts}(\theta)$  inductively, as follows. If  $\theta$  is an atom, or begins with disjunction or quantification, define  $\text{conjuncts}(\theta) = \{\theta\}$ , where  $\theta$  has multiplicity 1. When  $\theta$  has the form  $\theta_1 \wedge \theta_2$ , define  $\text{conjuncts}(\theta)$  as the multiset union  $\text{conjuncts}(\theta_1) \cup \text{conjuncts}(\theta_2)$ . We say that  $\theta$  is a *k-fold conjunction* when  $|\text{conjuncts}(\theta)| \geq k$ . We define *k-fold disjunctions* analogously.

The notion of *k-fold conjunction* will be useful in the remainder as it allows us to understand when we can apply the splitdown rule on quantifiers. The following lemma tells us that the existence of *k-fold conjunctions* in our equivalence classes with respect to  $\{A, C, O\}$  and  $\mathcal{T}$  can be decided by looking at any pushed-down representative.

► **Lemma 17.** *Suppose that  $\theta$  is a pfo-formula with  $[\theta]_{ACO}$  pushed-down, and let  $k \geq 1$ . Then, there exists a *k-fold conjunction* in  $[\theta]_{\mathcal{T}}$  if and only if  $\theta$  is a *k-fold conjunction*. Similarly, there exists a *k-fold disjunction* in  $[\theta]_{\mathcal{T}}$  if and only if  $\theta$  is a *k-fold disjunction*.*

As a consequence of Lemma 17, we get that for pushed-down formulas there is a result similar to Lemma 16 that characterizes when the pushdown rule can be applied.

► **Lemma 18.** *Let  $\theta$  be a pfo-formula where  $[\theta]_{ACO}$  is pushed-down. Then the following are equivalent:*

- $\rightarrow_{S\downarrow}$  can be applied to  $\theta$ .
- $\rightarrow_{S\downarrow}$  can be applied to  $[\theta]_{ACO}$ .
- $\rightarrow_{S\downarrow}$  can be applied to  $[\theta]_{\mathcal{T}}$ .

**Proof.** The implications from the first point to the second and from the second to the third are again directly clear as in the proof of Lemma 16.

So assume now that  $\rightarrow_{S\downarrow}$  can be applied to  $[\theta]_{\mathcal{T}}$ . Then, by definition, there exists a formula  $\theta^+ \in [\theta]_{\mathcal{T}}$  to which  $\rightarrow_{S\downarrow}$  can be applied. Since applicability of  $\rightarrow_{S\downarrow}$  to a formula and whether or not  $[\theta]_{ACO}$  is pushed-down are preserved by variable renaming, we may assume by Proposition 13 that  $\theta$  is standardized, and that  $\theta^+ \in [\theta]_{\mathcal{T} \setminus \{N\}}$ . Clearly, the formulas  $\theta$  and  $\theta^+$  are non-atomic. Observe that when  $\rightarrow_{S\downarrow}$  is applicable to an organized formula, the quantification must come from a holey  $\{\exists, \wedge\}$ -formula and the connective must come from a holey  $\{\forall, \vee\}$ -formula, or vice-versa.

By appeal to the decomposition of each non-atomic pfo-formula into an organized formula (Proposition 14), and Proposition 15, there exist organized formulas  $\phi[\psi_1, \dots, \psi_k]$ ,  $\phi^+[\psi_1^+, \dots, \psi_k^+]$  that are subformulas of  $\theta$  and  $\theta^+$ , respectively, where (1)  $\phi \xleftrightarrow{*}_{\mathcal{T} \setminus \{N\}} \phi^+$  and  $\psi_1 \xleftrightarrow{*}_{\mathcal{T} \setminus \{N\}} \psi_1^+, \dots, \psi_k \xleftrightarrow{*}_{\mathcal{T} \setminus \{N\}} \psi_k^+$ ; and (2)  $\rightarrow_{S\downarrow}$  can be applied to an instance of quantification in  $\phi^+$  and a connective in a formula  $\psi_\ell^+$ .

We assume up to duality that  $\phi^+$  is a holey  $\{\forall, \vee\}$ -formula and that  $\psi_\ell^+$  is a  $\{\exists, \wedge\}$ -organized formula. Viewing  $\phi^+$  and  $\psi_\ell^+$  as trees, we have that  $\ell$  occurs as a leaf in  $\phi^+$ , that the parent of this leaf in  $\phi^+$  is a universal quantification  $\forall x$ , and that  $\psi_\ell^+$  has conjunction ( $\wedge$ ) at its root. Thus  $\psi_\ell^+$  is a 2-fold conjunction; by Lemma 17, it follows that  $\psi_\ell$  is a 2-fold conjunction. Let  $B$  be the set of holes occurring in  $\phi$ ; since  $\phi \xleftrightarrow{*}_{\mathcal{T} \setminus \{N\}} \phi^+$ , we have that  $B$  is also the set of holes occurring in  $\phi^+$ . We have  $\text{free}(\psi_i) = \text{free}(\psi_i^+)$  for each  $i = 1, \dots, k$ . In applying rules in  $\mathcal{T} \setminus \{N\}$  to obtain  $\phi$  from  $\phi^+$ , which of the free occurrences of  $x$  from the

## 16:12 Optimally Rewriting Formulas and Database Queries

formulas  $\psi_i$  ( $i \in B$ ) are bound to the mentioned quantification  $\forall x$ , is preserved. Thus, among the formulas  $\psi_i$  ( $i \in B$ ), only  $\psi_\ell$  can have a free occurrence bound to the corresponding quantification  $\forall x$  in  $\phi$ . It thus follows that, in  $\phi$  viewed as a tree, there is no instance of disjunction between  $\forall x$  and  $\ell$ , for if there were, by quantifier reordering (applying  $\rightarrow_O$ ),  $\forall x$  could be moved above a disjunction, and then  $\rightarrow_{P\downarrow}$  would be applicable (contradicting that  $[\theta]_{ACO}$  is pushed-down). Thus, in  $\phi$  viewed as a tree, the parent of  $\ell$  is a universal quantification, and so  $\rightarrow_{S\downarrow}$  can be applied to  $\theta$ .  $\blacktriangleleft$

With the insights on rule applicability from above, we can now show that there is a polynomial time algorithm that decides if rules from the system  $Y = (\text{PFO}, \rightarrow_{\{P\downarrow, S\downarrow, M\}}) / \overset{*}{\leftrightarrow}_{ACO}$  can be applied to an equivalence class with respect to  $ACO$ . This will be a building block in the proof of Theorem 7 that shows that we can compute normal forms in the system  $Y$  in polynomial time.

**► Lemma 19.** *There is a polynomial time algorithm that, given a pfo-formula  $\theta$ , decides if there is a formula  $\theta' \in [\theta]_{ACO}$  such that a rule of  $Y$  can be applied to it. If so, it computes such a  $\theta'$  and a formula  $\theta''$  obtainable by applying a rule of  $Y$  to  $\theta'$ .*

**Proof.** We show this for the three rules in  $Y$  individually, starting with  $\rightarrow_M$ , then  $\rightarrow_{P\downarrow}$  and finally  $\rightarrow_{S\downarrow}$ .

For  $\rightarrow_M$ , by Lemma 16, a quantifier can be deleted in  $\theta$  if and only if one can be deleted from any  $\theta' \in [\theta]_{ACO}$ ; we thus directly get a polynomial-time algorithm for  $\rightarrow_M$ .

Now assume that  $\rightarrow_M$  cannot be applied, so every quantifier binds a variable in at least one atom in its subformula in  $\theta$ . Fix one quantifier  $v$  in  $\theta$ , say that quantifier is universal; the other case is totally analogous. Let  $\theta_v$  denote the subformula of  $\theta$  rooted in  $v$ . We assume that the quantifiers are checked inductively in a bottom-up fashion, so all quantifiers in strict subformulas of  $\theta_v$  have been checked before dealing with  $v$ . So we cannot apply  $\rightarrow_{P\downarrow}$  on any of them for any  $\theta' \in [\theta]_{ACO}$ . By Proposition 14, we have that the subformula  $\theta_v$  is organized. Let  $\psi$  be the region of  $\theta_v$  that  $v$  is applied on, i.e., the largest holey  $\{\forall, \vee\}$ -subformula in  $\theta$  containing the root of the subformula that  $v$  is applied on. If  $\psi$  only has one hole, then clearly we cannot apply  $\rightarrow_{P\downarrow}$  on  $v$  in  $\theta$  as then  $\psi$  does not contain any  $\vee$ -operation on which we could apply the pushdown. This is also true for all formulas  $\theta' \in [\theta]_{ACO}$  since the corresponding holey formula  $\psi'$  in  $\theta'$  also contains no  $\vee$ -operation, because the rules in  $\rightarrow_{ACO}$  cannot move them over other operators. So in particular, if we want to apply  $\rightarrow_{P\downarrow}$  to  $v$ , then  $\psi$  must contain a  $\vee$ -operation. After potentially applying  $\rightarrow_O$  several times, we may assume that the quantification  $v$  in  $\theta$  is applied on a disjunction. Then we can write  $\psi$  as an  $r$ -fold disjunction

$$\psi = \bigvee_{i=1}^r \psi_i, \tag{1}$$

for some integer  $r$  where the  $\psi_i$  are either holes or have a universal quantifier on top. Now if there is  $i^* \in [r]$  such that  $\psi_{i^*}$  does not contain  $x$  in any of its holes, we can use  $\rightarrow_A$  to rewrite  $\psi = \psi_{i^*} \vee \bigvee_{i \in [r] \setminus \{i^*\}} \psi_i$  which gives us an  $\vee$ -operation to which we can apply  $\rightarrow_{P\downarrow}$  for  $v$ . On the other hand, if all  $\psi_i$  contain  $x$  in one of their holes, then this is the case for all  $\psi'$  that we get for  $\theta'$ , because  $\rightarrow_{ACO}$  does not allow exchanging the positions of  $\vee$  and any other operators. Thus, the representation (1) for all  $\psi'$  is the same as for  $\psi$  up to fact that  $\rightarrow_{ACO}$  might have been used on the disjuncts  $\psi_i$ . But in that case, in no  $\psi'$  and thus in no  $\theta'$  there is a  $\vee$ -operation that  $v$  could be pushed over. This directly yields a polynomial time algorithm for this case.

Finally, assume that  $\rightarrow_M$  and  $\rightarrow_{P\downarrow}$  can both not be applied. Then in particular  $\theta$  is pushed-down. Then, by Lemma 18, we get that  $\rightarrow_{S\downarrow}$  can be applied to  $[\theta]_{ACO}$  if and only if it can be applied to  $\theta$  which directly yields a polynomial time algorithm.

Overall, we have polynomial time algorithms for all three rules of  $Y$ . Observing that if any rule can be applied, we can compute  $\theta'$  and the result  $\theta''$  of the rule application efficiently, completes the proof.  $\blacktriangleleft$

Finally, we give the proof of the normal form Theorem 8 which we restate for the convenience of the reader. Remember that  $Y' = (\text{PFO}, \rightarrow_{\{S\downarrow, M\}}) / \xrightarrow{*}_{\mathcal{T}}$ .

**► Theorem 8.** *Suppose that  $\phi$  is a pfo-formula where  $[\phi]_{ACO}$  is a normal form of the system  $Y$ ; then,  $[\phi]_{\mathcal{T}}$  is a normal form of the system  $Y'$ .*

**Proof.** Suppose that  $[\phi]_{ACO}$  is a normal form of the system  $Y$ . Then, we have that  $[\phi]_{ACO}$  is pushed-down, since  $P\downarrow$  is a rule in  $Y$ . Since  $[\phi]_{ACO}$  is a normal form of  $Y$ , neither of the rules  $P\downarrow$ ,  $M$  can be applied to  $[\phi]_{ACO}$ , implying by Lemmas 16 and 18 that neither of these two rules can be applied to  $[\phi]_{\mathcal{T}}$ . Thus  $[\phi]_{\mathcal{T}}$  is a normal form of  $Y'$ .  $\blacktriangleleft$

## 7 Termination and confluence

In this section, we will show that both systems  $Y$  and  $Y'$  are terminating. With Lemma 19 from the last section, this will yield Theorem 7, showing that we can efficiently compute normal forms for the system  $Y$ . For  $Y'$ , we will also show that it is locally confluent which then implies Theorem 9, the convergence of  $Y'$ .

We will start with termination for the system  $Y$ . Let us for every pfo-formula  $\phi$  denote by  $|\phi|$  the number of nodes in the syntax tree of  $\phi$ .

**► Lemma 20.** *For every pfo-formula  $\phi$ , any reduction chain in the system  $Y$  starting in  $[\phi]_{ACO}$  has length at most  $|\phi|^3$ .*

**Proof.** Consider a pfo-formula  $\phi$ . We will show that any chain of  $\rightarrow_{P\downarrow, S\downarrow, M}$ -steps starting in  $[\phi]_{ACO}$  is upper bounded by  $|\phi|^3$ . To this end, we define a potential function  $p$  on all pfo-formulas as follows: let  $F$  be a subformula of  $\phi$  whose root is labeled by a quantifier. Then the local potential  $\bar{p}(F)$  of  $F$  is defined as  $a^2$  where  $a$  is the number of atoms in  $F$ . Note that  $\bar{p}(F)$  is positive. Then the potential of  $\phi$  is the sum of the local potentials of all subformulas that have a quantifier labeling their root.

We claim that applying any operation in  $\rightarrow_{P\downarrow, S\downarrow, M}$  decreases the potential. So let  $\phi'$  be obtained from  $\phi$  by applying one reduction step. We consider the different possible cases:

- $\rightarrow_{P\downarrow}$ : Let  $F = \exists x(F_1 \wedge F_2)$  be a sub-formula of  $\phi$  such that in  $F_2$  the variable  $x$  is not free, and consider the application  $\exists x(F_1 \wedge F_2) \rightarrow_{P\downarrow} \underbrace{(\exists x F_1) \wedge F_2}_{:= F'}$ . The only change in the potential of  $\phi$  when applying this rule is the change from  $\bar{p}(F)$  to  $\bar{p}(F')$ , since no other sub-formulas of  $\phi$  change. But  $F_1$  contains fewer atoms than  $F_1 \wedge F_2$ , so  $\bar{p}(F) > \bar{p}(F')$ , so the potential of  $\phi$  decreases. All other cases for  $\rightarrow_{P\downarrow}$  follow analogously.
- $\rightarrow_{S\downarrow}$ : Let  $F = \forall x(F_1 \wedge F_2)$  be a sub-formula of  $\phi$  and consider the application  $\forall x(F_1 \wedge F_2) \rightarrow_{S\downarrow} (\forall x F_1) \wedge (\forall x F_2)$ . Then this application changes the potential  $p$  of  $\phi$  by  $\bar{p}(\forall x F_1) + \bar{p}(\forall x F_2) - \bar{p}(F)$ . Let  $a_1$  be the number of atoms in  $F_1$  and  $a_2$  the number of atoms in  $F_2$ , then

$$\bar{p}(\forall x F_1) + \bar{p}(\forall x F_2) = a_1^2 + a_2^2 < (a_1 + a_2)^2 = \bar{p}(F),$$

so the potential of  $F$  decreases. All other cases for  $\rightarrow_{S\downarrow}$  follow analogously.



## 16:14 Optimally Rewriting Formulas and Database Queries

- $\rightarrow_M$ : In the definition of  $p(F)$ , we lose one positive summand whenever applying  $\rightarrow_M$  without changing any of the other summands, so the potential decreases.

So in any case, whenever applying one of the rules on  $\phi$ , the potential  $p$  decreases.

Now consider the equivalence classes of  $PFO / \leftrightarrow_{ACO}^*$ . Note that applying  $\rightarrow_A$ ,  $\rightarrow_C$  or  $\rightarrow_O$  does not change the potential of any formula, so we can define the potential  $[p]$  of every equivalence class  $[\phi]_{ACO}$  by  $[p]([\phi]_{ACO}) := p(\phi)$ . Now, whenever applying a rule of the system to an equivalence class  $[\phi]_{ACO}$ , the potential  $[p]([\phi]_{ACO})$  decreases, because, as we have seen before, it decreases for any  $\phi' \in [\phi]_{ACO}$ .

The potential is bounded by  $[p]([\phi]_{ACO}) \leq |\phi|a^2 \leq |\phi|^3$  where  $|\phi|$  is the length of  $\phi$  and  $a$  the number of atoms. Since the potential is a positive integer, the longest descending chain starting in  $[\phi]_{ACO}$  is thus of length  $|\phi|^3$ , so any chain in the system has polynomially bounded length. ◀

The proof of Theorem 7 follows from this lemma straightforwardly.

We next turn to the termination of  $Y'$ , which will be used to show Theorem 9 with Lemma 2.

► **Lemma 21.** *The system  $Y'$  is terminating.*

**Proof sketch.** The proof follows the same idea as that of Lemma 20: we introduce a potential for every pfo-formula and show that it is the same for all representatives of an equivalence class. Then we show that applying rules from the system  $Y'$  decreases the potential of any formula. Unfortunately, the setting is slightly more complicated such that the definition of the potential is more involved than for Lemma 20. ◀

As the second ingredient for the proof of confluence for  $Y'$  with the help of Lemma 2, we now show local confluence of  $Y'$ .

► **Lemma 22.** *The system  $Y'$  is locally confluent.*

We now apply Lemma 2 using Lemma 21 and Lemma 22, to directly get Theorem 9.

## 8 Between rewriting and tree decompositions

The aim of this section is to link tree decompositions and formula width. We will then use our development to prove Theorem 11. In what follows, we focus on  $\{\exists, \wedge\}$ -formulas, but our results apply to the corresponding dual formulas, namely,  $\{\forall, \vee\}$ -formulas.

Let  $\phi$  be for now a standardized holey  $\{\exists, \wedge\}$ -formula, and let  $a$  be an association for  $\phi$ . We define the hypergraph of  $(\phi, a)$  as the hypergraph whose vertex set is  $\bigcup_i a(i)$ , where the union is over all holes  $i$  appearing in  $\phi$ , and whose edge set is  $\{a(i) \mid i \text{ appears in } \phi\} \cup \{\text{free}(\phi)\}$ ; that is, this hypergraph has an edge corresponding to each hole of  $\phi$ , and an edge made of the free variables of  $\phi$ .

The following theorem identifies a correspondence between the rules in  $\mathcal{T} \setminus \{N\}$  and the computation of minimum tree decompositions. A related result appears as [13, Theorem 7].<sup>3</sup>

► **Theorem 23.** *Let  $\phi$  be a standardized holey  $\{\exists, \wedge\}$ -formula, and let  $a$  be an association for  $\phi$ . Let  $H$  be the hypergraph of  $(\phi, a)$ . It holds that  $\text{width}([\phi]_{\mathcal{T} \setminus \{N\}}) = \text{tw}(H) + 1$ . Moreover, there exists a polynomial-time algorithm that, given the pair  $\phi, a$  and a tree decomposition  $C$  of  $H$ , outputs a formula  $\phi' \in [\phi]_{\mathcal{T} \setminus \{N\}}$  where  $\text{width}(\phi') \leq \text{bagsize}(C)$ ; when  $C$  is a minimum width tree decomposition of  $H$ , it holds that  $\text{width}(\phi') = \text{bagsize}(C)$ .*

<sup>3</sup> Let us remark that, in the context of the present theorem, the presence of the rewriting rule  $O$  is crucial: Example 4 shows that absence of this rule affects the minimum width achievable.



The following is a consequence of Theorem 23, and Proposition 13.

► **Corollary 24.** *Let  $\phi$  be a standardized  $\{\exists, \wedge\}$ -sentence, and let  $H$  be the hypergraph of  $\phi$  (defined as having a hyperedge  $\{v_1, \dots, v_k\}$  for each atom  $R(v_1, \dots, v_k)$  of  $\phi$ ). It holds that*

$$\text{width}([\phi]_{\mathcal{A}}) = \text{width}([\phi]_{\mathcal{T}}) = \text{width}([\phi]_{\mathcal{T} \setminus \{N\}}) = \text{tw}(H) + 1.$$

Let us briefly explain how to employ Theorem 23 to prove Theorem 11. The proof relies on the fact that, via Proposition 15, we can consider the regions of the formula  $\theta$  independently. Moreover, Theorem 23 allows us to minimize the width of the individual regions.

## 9 Monotonicity

In this section, we prove monotonicity of the gauged system  $G'$ . We begin by proving a theorem concerning the width, up to rules in  $\mathcal{T} \setminus \{N\}$ , of holey  $\{\exists, \wedge\}$ -formulas that are conjunctive. This will allow us to understand how to use these rules to minimize a formula that permits application of the splitdown rule.

► **Theorem 25.** *Let  $\theta$  be a standardized holey  $\{\exists, \wedge\}$ -formula having the form  $\phi \wedge \phi'$ , and let  $a$  be an association for  $\theta$ . It holds that  $\text{width}([\phi \wedge \phi']_{\mathcal{T} \setminus \{N\}})$  is equal to*

$$\max(\text{width}([\phi]_{\mathcal{T} \setminus \{N\}}), \text{width}([\phi']_{\mathcal{T} \setminus \{N\}}), |\text{free}(\phi) \cup \text{free}(\phi')|).$$

Theorem 25 is used to show Theorem 10.

## 10 Conclusion

We have seen that one can optimally minimize the width of positive first-order formulas with respect to the application of syntactic rules. The algorithm we have given runs in polynomial time, up to the use of an algorithm computing minimum tree decompositions of hypergraphs. While the computation of such decompositions is known to be NP-hard, there exist FPT-algorithms parameterized by the treewidth (see for example [6, 5]) and it follows directly that our width minimization algorithm runs in FPT-time parameterized by the optimal width.

From our techniques, it can be seen that equivalence up to the syntactical rules that we study can also be checked for pfo-formulas: given two pfo-formulas  $\phi_1, \phi_2$ , compute their normal forms with the algorithm of Theorem 11. Then,  $\phi_1$  and  $\phi_2$  are equivalent if there is an isomorphism of the structure of their regions – which is easy to check since the regions are organized in a tree shape – and the regions mapped onto each other by this isomorphism are equivalent. The latter can be verified by standard techniques for conjunctive queries [9] which yields the algorithm for checking equivalence of pfo-formulas under our syntactical rules and, in the positive case, also allows extracting a sequence of rule applications that transforms  $\phi_1$  to  $\phi_2$ .

We close by discussing an open issue: it would be interesting to extend the set of rules we allow in the width minimization. A particularly natural addition would be the distributive rules stating that  $\wedge$  distributes over  $\vee$ , and vice-versa. Note that allowing this operation would necessarily change the form of the results we could hope for: it is known that there are formulas whose width minimization leads to an unavoidable exponential blow-up in the formula size [4], and it is verifiable that this is also true when only allowing syntactical rewriting rules including distributivity. Note that this is very different from our setting where the rewriting rules that we consider may only increase the formula size by adding a

polynomial number of quantifiers by the splitdown rule. Thus, when adding distributivity to our rule set, we cannot hope for polynomial time algorithms up to treewidth computation, as we showed in the present article. Still, it would be interesting to understand how distributivity changes the rewriting process beyond this and if there are algorithms that run polynomially, say, in the input and output size.

---

## References

- 1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995. URL: <http://webdam.inria.fr/Alice/>.
- 2 Isolde Adler and Mark Weyer. Tree-width for first order formulae. *Log. Methods Comput. Sci.*, 8(1), 2012. doi:10.2168/LMCS-8(1:32)2012.
- 3 Franz Baader and Tobias Nipkow. *Term rewriting and all that*. Cambridge University Press, 1998.
- 4 Christoph Berkholz and Hubie Chen. Compiling existential positive queries to bounded-variable fragments. In Dan Suciu, Sebastian Skritek, and Christoph Koch, editors, *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, pages 353–364. ACM, 2019. doi:10.1145/3294052.3319693.
- 5 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996. doi:10.1137/S0097539793251219.
- 6 Hans L. Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michal Pilipczuk. An  $o(c^k n)$  5-approximation algorithm for treewidth. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 499–508. IEEE Computer Society, 2013. doi:10.1109/FOCS.2013.60.
- 7 Simone Bova and Hubie Chen. The complexity of width minimization for existential positive queries. In Nicole Schweikardt, Vassilis Christophides, and Vincent Leroy, editors, *Proc. 17th International Conference on Database Theory (ICDT), Athens, Greece, March 24-28, 2014*, pages 235–244. OpenProceedings.org, 2014. doi:10.5441/002/icdt.2014.25.
- 8 Simone Bova and Hubie Chen. How many variables are needed to express an existential positive query? *Theory Comput. Syst.*, 63(7):1573–1594, 2019. doi:10.1007/s00224-018-9884-z.
- 9 Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In John E. Hopcroft, Emily P. Friedman, and Michael A. Harrison, editors, *Proceedings of the 9th Annual ACM Symposium on Theory of Computing, May 4-6, 1977, Boulder, Colorado, USA*, pages 77–90. ACM, 1977. doi:10.1145/800105.803397.
- 10 Hubie Chen. On the complexity of existential positive queries. *ACM Trans. Comput. Log.*, 15(1):9:1–9:20, 2014. doi:10.1145/2559946.
- 11 Hubie Chen. The tractability frontier of graph-like first-order query sets. *J. ACM*, 64(4):26:1–26:29, 2017. doi:10.1145/3073409.
- 12 Hubie Chen and Víctor Dalmau. Decomposing quantified conjunctive (or disjunctive) formulas. *SIAM J. Comput.*, 45(6):2066–2086, 2016. doi:10.1137/140957378.
- 13 Víctor Dalmau, Phokion G. Kolaitis, and Moshe Y. Vardi. Constraint satisfaction, bounded treewidth, and finite-variable logics. In Pascal Van Hentenryck, editor, *Principles and Practice of Constraint Programming - CP 2002, 8th International Conference, CP 2002, Ithaca, NY, USA, September 9-13, 2002, Proceedings*, volume 2470 of *Lecture Notes in Computer Science*, pages 310–326. Springer, 2002. doi:10.1007/3-540-46135-3\_21.
- 14 Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions: A survey. In Jirí Sgall, Ales Pultr, and Petr Kolman, editors, *Mathematical Foundations of Computer Science 2001, 26th International Symposium, MFCS 2001 Mariánské Lázně, Czech Republic, August 27-31, 2001, Proceedings*, volume 2136 of *Lecture Notes in Computer Science*, pages 37–57. Springer, 2001. doi:10.1007/3-540-44683-4\_5.



- 15 Georg Gottlob, Nicola Leone, and Francesco Scarcello. Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width. *Journal of Computer and System Sciences*, 66(4):775–808, 2003. doi:10.1016/S0022-0000(03)00030-8.
- 16 Martin Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *J. ACM*, 54(1):1:1–1:24, 2007. doi:10.1145/1206035.1206036.
- 17 Martin Grohe, Thomas Schwentick, and Luc Segoufin. When is the evaluation of conjunctive queries tractable? In Jeffrey Scott Vitter, Paul G. Spirakis, and Mihalis Yannakakis, editors, *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 657–666. ACM, 2001. doi:10.1145/380752.380867.
- 18 Phokion G. Kolaitis and Moshe Y. Vardi. Conjunctive-query containment and constraint satisfaction. *J. Comput. Syst. Sci.*, 61(2):302–332, 2000. doi:10.1006/jcss.2000.1713.
- 19 Maxwell Herman Alexander Newman. On theories with a combinatorial definition of “equivalence”. *Annals of mathematics*, pages 223–243, 1942.
- 20 Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.
- 21 Ryan Williams. Faster decision of first-order graph properties. In *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14*, pages 80:1–80:6. ACM, 2014. doi:10.1145/2603088.2603121.



# Containment of Regular Path Queries Under Path Constraints

Sylvain Salvati  

Université de Lille, INRIA, CRISTAL/CNRS UMR 9189, France

Sophie Tison  

Université de Lille, INRIA, CRISTAL/CNRS UMR 9189, France

---

## Abstract

---

Data integrity is ensured by expressing constraints it should satisfy. One can also view constraints as data properties and take advantage of them for several tasks such as reasoning about data or accelerating query processing. In the context of graph databases, simple constraints can be expressed by means of path constraints while simple queries are modeled as regular path queries (RPQs). In this paper, we investigate the containment of RPQs under path constraints. We focus on word constraints that can be viewed as tuple-generating dependencies (TGDs) of the form

$$\forall x_1, x_2, \exists \bar{y}, a_1(x_1, y_1) \wedge \dots \wedge a_i(y_{i-1}, y_i) \wedge \dots \wedge a_n(y_{n-1}, x_2) \longrightarrow \\ \exists \bar{z}, b_1(x_1, z_1) \wedge \dots \wedge b_i(z_{i-1}, z_i) \wedge \dots \wedge b_m(z_{m-1}, x_2) .$$

Such a constraint means that whenever two nodes in a graph are connected by a path labeled  $a_1 \dots a_n$ , there is also a path labeled  $b_1 \dots b_m$  that connects them. Rewrite systems offer an abstract view of these TGDs: the rewrite rule  $a_1 \dots a_n \rightarrow b_1 \dots b_m$  represents the previous constraint. A set of constraints  $\mathcal{C}$  is then represented by a rewrite system  $R$  and, when dealing with possibly infinite databases, a path query  $p$  is contained in a path query  $q$  under the constraints  $\mathcal{C}$  iff  $p$  rewrites to  $q$  with  $R$ . Contrary to what has been claimed in the literature we show that, when restricting to finite databases only, there are cases where a path query  $p$  is contained in a path query  $q$  under the constraints  $\mathcal{C}$  while  $p$  does not rewrite to  $q$  with  $R$ . More generally, we study the finite controllability of the containment of RPQs under word constraints, that is when this containment problem on unrestricted databases does coincide with the finite case. We give an exact characterisation of the cases where this equivalence holds. We then deduce the undecidability of the containment problem in the finite case even when RPQs are restricted to word queries. We prove several properties related to finite controllability, and in particular that it is undecidable. We also exhibit some classes of word constraints that ensure the finite controllability and the decidability of the containment problem.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Database constraints theory; Theory of computation  $\rightarrow$  Rewrite systems; Theory of computation  $\rightarrow$  Regular languages; Theory of computation  $\rightarrow$  Grammars and context-free languages; Theory of computation  $\rightarrow$  Database theory

**Keywords and phrases** Graph databases, rational path queries, query containment, TGDs, word constraints, rewrite systems, finite controllability, decision problems

**Digital Object Identifier** 10.4230/LIPIcs.ICDT.2024.17

**Funding** This work was financially supported by the ANR project CQFD (ANR-18-CE23-0003).

**Acknowledgements** We thank anonymous reviewers and Diego Figueira for their useful comments and Pierre Bourhis and Lily Gallois for fruitful discussions.

## 1 Introduction

**The problem.** In this paper, we investigate the containment of regular path queries (RPQs) under path constraints. We model graph databases as finite edge-labeled graphs. We call  $\omega$ -graph database (or  $\omega$ -database) graph databases where we remove the finiteness constraint. Queries we consider here are RPQs that test whether two nodes of the graph are connected



© Sylvain Salvati and Sophie Tison;  
licensed under Creative Commons License CC-BY 4.0  
27th International Conference on Database Theory (ICDT 2024).

Editors: Graham Cormode and Michael Shekelyan; Article No. 17; pp. 17:1–17:19

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

by a path whose label belongs to a given regular language. Query containment and query equivalence are important properties when dealing with data: they play a central role in query optimizations, and also in reasoning about data. Query containment for RPQs without constraints is simply the problem of regular languages containment. In practice, query containment is also often used when dealing with particular databases for which we have knowledge about the actual data. We focus here on knowledge expressed by *word constraints* of the form  $a_1 \dots a_n \sqsubseteq b_1 \dots b_m$ . Such a constraint means that whenever two nodes in a graph are connected by a path labeled  $a_1 \dots a_n$ , there is also a path labeled  $b_1 \dots b_m$  that connects them. From the logical point of view, this constraint can be seen as the following tuple-generating dependency (TGD):

$$\begin{aligned} \forall x_1, x_2, \exists \bar{y}, a_1(x_1, y_1) \wedge \dots \wedge a_i(y_{i-1}, y_i) \wedge \dots \wedge a_n(y_{n-1}, x_2) \longrightarrow \\ \exists \bar{z}, b_1(x_1, z_1) \wedge \dots \wedge b_i(z_{i-1}, z_i) \wedge \dots \wedge b_m(z_{m-1}, x_2) . \end{aligned}$$

When  $b_1 \dots b_m$  is the empty word, the constraint  $a_1 \dots a_n \sqsubseteq \varepsilon$  is actually an Equality-Generating Dependency (EGD) which can be written:

$$\forall x_1, x_2, \exists \bar{y}, a_1(x_1, y_1) \wedge \dots \wedge a_i(y_{i-1}, y_i) \wedge a_n(y_{n-1}, x_2) \longrightarrow x_1 = x_2 .$$

In the rest of the paper we are going to be concerned only by word constraints  $p \sqsubseteq q$  where neither  $p$  nor  $q$  is the empty string. EGDs pose problems slightly different from TGDs and part of the results of the paper does not apply when we remove this hypothesis.

Given a set of word constraints  $\mathcal{C}$  and two RPQs  $P$  and  $Q$ , we may wonder whether for every  $\omega$ -database that satisfies  $\mathcal{C}$  the answer set of the query  $P$  is included in that of  $Q$ . In this case we write  $P \sqsubseteq_{\mathcal{C}}^{\omega} Q$ . If we restrict our attention to (finite) databases, we then write  $P \sqsubseteq_{\mathcal{C}}^f Q$ . In the context of databases without any constraints, the query containment problem boils down to language inclusion and the relation of query containment coincides in the finite case and in the infinite case. Such properties are called *finitely controllable*.

Word constraints are able to define precisely complex notions and then ensure that they are well used in databases. A simple example consists in defining what it means to be of the same generation in a genealogical tree, i.e. connecting two persons that are at the same distance of a common ancestor. We assume that we are given the edge labels **child** (that connects a person  $x$  to a person  $y$  when  $x$  is the child of  $y$ ), **parent** (that connects a person  $x$  to a person  $y$  when  $x$  is the parent of  $y$ ) and **sg** (for *same generation*), the following constraints give a definition of the relation **sg**:

$$\begin{aligned} \text{child parent} &\subseteq \text{sg} \\ \text{child sg parent} &\subseteq \text{sg} \end{aligned}$$

Word constraints are basic and more refined properties would require more logical connective, e.g. modalities, joins on paths etc. However, the undecidability results of the paper for this basic class of constraints apply to more involved and more expressive classes.

**Rewrite systems and word constraints.** Suppose that there is a path labeled  $p_1 p p_2$  in an  $\omega$ -database that satisfies the constraint  $p \sqsubseteq q$ . We then know that there is a path  $p_1 q p_2$  that connects the two vertices in the  $\omega$ -database. If we further know that  $p_1 p p_2 \sqsubseteq q'$ , we can deduce that there exists a path  $q'$  between the two vertices. We can then apply the same kind of reasoning any number of times. This deduction mechanism is similar to a well known model of computation: *rewrite systems* or *semi-Thue systems*. Rewrite systems offer an abstract view of these particular TGDs: the rewrite rule  $a_1 \dots a_n \rightarrow b_1 \dots b_m$  can be associated with the constraint  $a_1 \dots a_n \sqsubseteq b_1 \dots b_m$ . A (finite) set of constraints  $\mathcal{C}$  is then represented by a (finite) rewrite system  $R$ .

Given  $\mathcal{C}$  a finite set of word constraints, consider the containment problem  $u \sqsubseteq_{\mathcal{C}}^{\omega} v$  where  $u$  and  $v$  are words. It is easy to see that if  $u$  rewrites to  $v$ , we have  $u \sqsubseteq_{\mathcal{C}}^{\omega} v$ , as we have seen that  $\sqsubseteq_{\mathcal{C}}^{\omega}$  is transitive and closed under context (i.e.  $u \sqsubseteq_{\mathcal{C}}^{\omega} v$  implies  $w_1 u w_2 \sqsubseteq_{\mathcal{C}}^{\omega} w_1 v w_2$ ). With a classic construction, we can build an infinite model  $D$  of  $\mathcal{C}$  such that  $D \models u \sqsubseteq v$  iff  $u$  rewrites to  $v$  (see Theorem 2). So, for word constraints,  $\sqsubseteq_{\mathcal{C}}^{\omega}$  coincides with the associated rewrite relation and so, it is undecidable as the word problem (deciding whether two words are in the rewrite relation) for rewrite systems is in general undecidable.

Word constraints in databases and rewrite systems have already been connected in different frameworks, e.g. for *rooted* path constraints in *rooted* databases [1, 8] and for constraints in document stores [5]. The framework we consider here is the same as in [14] that emphasizes this strong connection between the query containment problem in the presence of word constraints and the word problem in rewrite systems. However, the connection is slightly more subtle than expected and we investigate it.

Whereas  $\sqsubseteq_{\mathcal{C}}^{\omega}$  and  $\xrightarrow{*}_{\mathcal{C}}$  coincide,  $\sqsubseteq_{\mathcal{C}}^f$  and  $\xrightarrow{*}_{\mathcal{C}}$  do not coincide in general - contrary to what is stated in [14, Theorem 2]. Indeed, on the one hand, the set of descendants of  $u$  by  $\xrightarrow{*}_{\mathcal{C}}$  is recursively enumerable. On the other hand, the set  $\{v \mid u \sqsubseteq_{\mathcal{C}}^f v\}$  is co-recursively enumerable [8]: one can enumerate the databases until one finds  $D$  so that  $D \models \mathcal{C}$  and there is a path labeled  $u$  between two nodes and no path labeled  $v$  between these two nodes. So, if they coincide, they are recursive: as soon as the set of descendants of  $u$  by  $\xrightarrow{*}_{\mathcal{C}}$  is not recursive, it cannot be the case that the two sets coincide. As a consequence, *there must be cases where  $u \sqsubseteq_{\mathcal{C}}^f v$  while it is not true that  $u \xrightarrow{*}_{\mathcal{C}} v$* . We exhibit concrete examples that illustrate this phenomenon in the paper.

**Query containment is not finitely controllable.** By the preceding remark, in the setting of word constraints, for RPQs,  $\sqsubseteq_{\mathcal{C}}^{\omega}$  and  $\sqsubseteq_{\mathcal{C}}^f$  do not coincide: *query containment is not finitely controllable*. This result is central in this paper.

We study the finite controllability of the containment of RPQs under word constraints. We give an exact characterization of  $P \sqsubseteq_{\mathcal{C}}^f Q$  relying on  $\xrightarrow{*}_{\mathcal{C}}$ . More precisely,  $P \sqsubseteq_{\mathcal{C}}^f Q$  holds iff every regular language closed under  $R$  that intersects with  $P$  intersects with  $Q$ . We then deduce from this characterization the undecidability of the containment problem in the finite case even when RPQs are restricted to word queries. This characterization also allows us to better understand when this containment problem on unrestricted databases does coincide with the finite case. We investigate several aspects of the finite controllability, and, in particular, we prove its undecidability. We also exhibit some classes of word constraints that ensure the finite controllability and the decidability of the containment problem.

**Related work.** As we already pointed out, this paper is strongly related to [14]. The setting we consider is the same. We correct some false claims -mainly the finite controllability of the query containment problem- announced in the paper and give new proofs of correct results whose proofs were relying on the finite controllability of the query containment problem. If the proofs are new ones, some ideas in our proofs were already present in [14].

The strong link between rewriting and path constraints has been investigated in [1, 3] in the rooted case: graphs are rooted, and queries are always evaluated starting from the root. In this setting, this amounts to use rewrite system with the *prefix rewriting strategy*, i.e. if  $u \rightarrow v$  is a rule, only rewritings of the form “ $up$  rewrites to  $vp$ ” are allowed. Prefix rewriting preserves regularity [7], and given a word  $u$ , it is easy to build a finite database with two nodes  $n_1, n_2$ , such that there is a path  $v$  between  $n_1$  and  $n_2$  iff  $v$  is a descendant of  $u$  by this prefix rewriting. This ensures the finite controllability of query containment and the decidability of



regular path queries containment in the rooted case. This construction cannot be extended in general to the non-rooted case: indeed, the language of paths between two nodes in a finite model is a regular language, so cannot coincide with the set of descendants of  $u$ , if this set is non regular. Preservation of regularity is a key property, even if we prove that this is sufficient but not necessary to guarantee finite controllability in our setting. Links between rewriting and path constraints have also been used in the context of ontology-mediated query answering [5] and consistent query answering [4].

Undecidability of path constraint implication has already been proved in different contexts. In particular, undecidability of (resp. finite) implication has been proved r.e. (resp. co-r.e.) complete in the context of rooted graphs for a constraint language allowing constraints of the form  $\forall x, (\alpha(r, x) \implies \forall y(\beta(x, y) \implies \gamma(x, y)))$  where  $r$  is a root of the graph,  $\alpha, \beta, \gamma$  are paths [8]. In [9], similar constraints are expressed in Description Logic (DL) and finite implication is proved undecidable both in the rooted and in the global semantics. The global semantics is actually more general than our setting. The undecidability result concerning the global semantics of [9, Theorem 15] is strong enough to prove the undecidability of the query containment problem. For the sake of completeness, we give an original and direct proof of this result in Section 5.

Finite controllability for containment of conjunctive queries under inclusion and functional dependencies was introduced in [16]. The notion of finite controllability was later studied in several papers. In particular, finite controllability of containment for conjunctive queries under arbitrary inclusion dependencies and under keys and foreign keys has been proved in [18, 19] and finite controllability of UCQs was later showed for several classes of constraints, e.g. [12, 13, 2]. Consistent query answering for CRPQs under conjunction regular-path constraints have been studied in [4]. Finite Controllability for Ontology-Mediated Query Answering of C2RPQs has been studied in [10] where a complete classification of fragments of C2RPQs w.r.t. finite controllability under different classes of constraints, is provided according to the class of the underlying graph structure underlying the query. The results we obtain here for finite controllability are disjoint from these results, as we restrict to word constraints and as we focus to RPQs containment. Let us note that the classes of word constraints ensuring finite controllability that we exhibit don't fall, as far as we know, in any of the classes identified as ensuring finite controllability of CQs in the literature.

**The problems we consider.** Here follow the definitions of the main decision problems at the heart of this paper:

<i>QC</i>	<b>Query Containment</b>
<i>Input</i>	A set of word constraints $\mathcal{C}$ , two RPQs $P, Q$
<i>Question</i>	$P \sqsubseteq_{\mathcal{C}}^f Q$ ?
<i>QC<sup>ω</sup></i>	<b>ω-Query Containment</b>
<i>Input</i>	A set of word constraints $\mathcal{C}$ , two RPQs $P, Q$
<i>Question</i>	$P \sqsubseteq_{\mathcal{C}}^{\omega} Q$ ?
<i>UFC</i>	<b>Uniform Finite Controllability</b>
<i>Input</i>	A set of word constraints $\mathcal{C}$
<i>Question</i>	For any RPQs $P, Q$ , $P \sqsubseteq_{\mathcal{C}}^f Q$ iff $P \sqsubseteq_{\mathcal{C}}^{\omega} Q$ ?
<i>FC</i>	<b>Finite Controllability</b>
<i>Input</i>	A set of word constraints $\mathcal{C}$ , two RPQs $P, Q$
<i>Question</i>	$P \sqsubseteq_{\mathcal{C}}^f Q$ iff $P \sqsubseteq_{\mathcal{C}}^{\omega} Q$ ?

Unfortunately, we will see that these problems are undecidable in general. So, we also consider subclasses of word constraints. Given *Problem* in  $\{QC, QC^{\omega}, UFC, FC\}$ , *Problem*( $\mathcal{C}$ ) will denote *Problem* restricted to the class  $\mathcal{C}$  of word constraints. These

■ **Table 1** Decidability of query containment under word constraints (U: undecidable, D: decidable).

Decision Problem	Word Constraints $\mathcal{C}$				
	Unrestricted	Recursive	$\mathcal{C}$ preserves effectively regularity	$\mathcal{C}^{-1}$ preserves effectively regularity	Class of Proposition 26
$wwQC^\omega$	U	D	D	D	U
$QC^\omega$	U	U	U	D	U
$wwQC$	U	?	D	D	D
$QC$	U	U	U	D	D

■ **Table 2** Finite controllability of query containment under word constraints (U: undecidable).

Decision Problem	Word Constraints $\mathcal{C}$				
	Unrestricted	$\mathcal{C}$ preserves effectively regularity	$\mathcal{C}^{-1}$ preserves effectively regularity	Class of Proposition 26	Example of Theorem 13
$wwUFC$	U	Yes	Yes	No	Yes
$wuUFC$	U	Yes	Yes	No	No

problems can also be restricted to the cases when  $P$  or  $Q$  is a word query. We denote by  $XYProblem(\mathcal{C})$  with  $X, Y \in \{U, W\}$ , the problem  $Problem(\mathcal{C})$  where  $P$  (resp.  $Q$ ) is restricted to words if  $X = W$  (resp.  $Y = W$ ), unrestricted if  $X = U$  (resp.  $Y = U$ ). When both  $X$  and  $Y$  are  $U$ , they can be omitted. If  $\mathcal{C}$  corresponds to the whole class of word constraints, it can be omitted. E.g.  $UWQC$  denotes the problem of query containment where  $P$  is an RPQ and  $Q$  is a word; let  $\mathcal{CF}$  the class of word constraints associated with context-free grammars,  $wwQC(\mathcal{CF})$  denotes the problem of query containment where  $P$  and  $Q$  are words and  $R$  belongs to  $\mathcal{CF}$ .

In the sequel, when we mention *query containment* we mean  $QC$ , i.e. we refer to the *finite case*.

**Summary of the results.** Table 1 and Table 2 summarize most of the results presented in the paper.

## 2 Preliminaries

**Graph databases.** We model graph databases as edge-labeled graphs. For this we fix a finite alphabet of *labels*  $\Sigma$ , a *graph database* (or *database*)  $D$  is a pair  $(V, E)$  where  $V$  is a finite set of *objects* and  $E \subseteq V \times \Sigma \times V$  is a finite set of *directed labeled edges*. We call  $\omega$ -*graph database* (or  $\omega$ -*database*) graph databases where we remove the finiteness constraint. When there is an edge  $(x, a, y)$  in an  $\omega$ -database we say that there is an *edge labeled  $a$  between  $x$  and  $y$*  or *from  $x$  to  $y$* . As it is customary, we allow ourselves to write  $a(x, y)$  for the edge  $(x, a, y)$ . Finally, we abuse notation and write  $x \in D$  or say  $x$  is in  $D$  to mention that  $x$  is an object of  $D$ . Similarly, for edges, we write  $a(x, y) \in D$  or say  $a(x, y)$  belongs to  $D$ .

**Path queries and database constraints.** The *set of paths of labels* (or simply *paths*) over the alphabet  $\Sigma$  is  $\Sigma^*$  the set of (possibly empty) words or sequences built on  $\Sigma$ . We write  $\varepsilon$  for the empty path and  $\Sigma^+$  for  $\Sigma^* \setminus \{\varepsilon\}$ . We inductively define the fact that two objects  $x$  and  $y$  of a  $\omega$ -database  $D$  are connected by a path labeled  $u$  (we note this property  $u(x, y) \in D$ ) as follows:

- if  $u = \varepsilon$ , then  $u(x, y) \in D$  iff  $x = y$ ,
- if  $u = va$ , then  $u(x, y) \in D$  iff there is  $z$  so that  $v(x, z) \in D$  and  $a(z, y) \in D$ .

A path labeled  $u$  is considered as a *query* whose answer on an  $\omega$ -database  $D$  is

$$\text{ans}(u, D) = \{(x, y) \mid u(x, y) \in D\} .$$

Adopting the logical point of view, a path query  $u = a_1 \cdots a_n$  can be seen as a particular kind of conjunctive query:

$$\exists x_1. \dots \exists x_{n-1}. a_1(x, x_1) \wedge a_2(x_1, x_2) \wedge \dots \wedge a_n(x_{n-1}, y) .$$

In actual graph database systems, queries do not restrict to path labels but rather to *path labels languages*. Subsets of  $\Sigma^*$  are called *languages*. A language  $Q$  can also be seen as a query. The answer to that query on the  $\omega$ -database  $D$  is:

$$\text{ans}(Q, D) = \{(x, y) \mid \exists u \in Q. u(x, y) \in D\} .$$

In other words, such a query collects all the ordered pairs of nodes that are connected by a path labeled in  $Q$ . When  $Q$  is a regular language, the query induced by  $Q$  is called *regular path query* (RPQ).

Given two languages  $P$  and  $Q$  included in  $\Sigma^*$  and an  $\omega$ -database  $D$ , whenever  $\text{ans}(P, D) \subseteq \text{ans}(Q, D)$  we say that  $D$  satisfies the *constraint*  $P \sqsubseteq Q$  which we denote with  $D \models P \sqsubseteq Q$ . An  $\omega$ -database  $D$  satisfies a set constraints  $\mathcal{C}$  when for every  $P \sqsubseteq Q \in \mathcal{C}$ ,  $D \models P \sqsubseteq Q$ ; in that case we write  $D \models \mathcal{C}$ . When for every  $\omega$ -database  $D$ ,  $D \models \mathcal{C}$  implies  $D \models P \sqsubseteq Q$ , we write  $P \sqsubseteq_{\mathcal{C}}^{\omega} Q$ . When for every (finite) database  $D$ ,  $D \models \mathcal{C}$  implies  $D \models P \sqsubseteq Q$ , we write  $P \sqsubseteq_{\mathcal{C}}^f Q$ . Clearly,  $P \sqsubseteq_{\mathcal{C}}^{\omega} Q$  implies  $P \sqsubseteq_{\mathcal{C}}^f Q$ . In this paper, we focus on finite sets of *word constraints*, i.e. constraints of the form  $\{p\} \sqsubseteq \{q\}$  where  $p \neq \varepsilon$  and  $q \neq \varepsilon$ , that we also write  $p \sqsubseteq q$ . We also focus on properties  $P \sqsubseteq_{\mathcal{C}}^f Q$  and  $P \sqsubseteq_{\mathcal{C}}^{\omega} Q$  when  $P$  and  $Q$  are regular languages that do not contain  $\varepsilon$  and sometimes more specifically when  $P$  or  $Q$  are singleton sets, i.e. represent words.

**Rewrite systems.** A rewrite system  $R$  on an alphabet  $\Sigma$  is a finite set of rules of the form  $u \rightarrow v$  with  $u, v$  in  $\Sigma^+$ . The *one-step rewrite relation* of  $R$ , noted  $\rightarrow_R$ , is defined as follows:  $p \rightarrow_R q$  when  $p = u_1 u u_2$ ,  $q = u_1 v u_2$  and  $R$  contains the rule  $u \rightarrow v$ . We note  $\xrightarrow{*}_R$  the reflexive transitive closure of  $\rightarrow_R$ . We write  $D_R(u)$  for the set of *descendants* of  $u$ ,  $\{v \mid u \xrightarrow{*}_R v\}$ . For a language  $L$ ,  $D_R(L)$  is  $\bigcup_{u \in L} D_R(u)$ . We similarly define the set  $A_R(u)$  of *ancestors* of  $u$ ,  $\{v \mid v \xrightarrow{*}_R u\}$  and  $A_R(L) = \bigcup_{u \in L} A_R(u)$ . A language  $L$  is *closed under*  $R$  when  $D_R(L) = L$ . We let  $R^{-1}$  be the rewrite system obtained by reversing each rule of  $R$  ( $A_R(L) = D_{R^{-1}}(L)$ ).

We restrict rewrite systems to rules with non-empty words as we only wish to consider word constraints that are representable by means of TGDs. Notice that this restriction does not diminish the *computational power* of rewrite systems.

The *word problem* for rewrite systems is the question, given two words  $u$  and  $v$ , whether  $u \xrightarrow{*}_R v$ . This question is known to be undecidable, even with rules with non-empty words. We denote the set of left-hand (right-hand) sides of the rules in  $R$  by  $lhs(R)$  ( $rhs(R)$ ). We will use the following “modularity” property whose proof can be found in Appendix A:

► **Lemma 1.** *Let  $R = R_1 \cup R_2$  a rewrite system such that the letters occurring in  $\text{lhs}(R_1)$  do not occur in  $\text{rhs}(R_2)$ , then,  $\overset{*}{\rightarrow}_R = \overset{*}{\rightarrow}_{R_1} \circ \overset{*}{\rightarrow}_{R_2}$*

**Grammars.** We will also use particular types of rewrite systems: grammars. A *type-0 grammar*  $G$  is a tuple  $(N, \Sigma, S, R)$  where  $N$  is a finite set of *non-terminals*,  $\Sigma$  is a finite set of *terminals*,  $S$  is an element of  $N$ , *the axiom of  $G$* , and  $R$  is a finite set of rewrite rules on the alphabet  $N \cup \Sigma$ . The language defined by  $G$  is  $\mathcal{L}(G) = \{w \in \Sigma^* \mid S \overset{*}{\rightarrow}_R w\}$ . Notice that the rules of grammars being rewrite rules, they use non-empty words. It is well known that every recursively enumerable language can be defined by means of type-0 grammars. If  $L$  is a recursive language in  $\Sigma^+$ ,  $L$  and its complement in  $\Sigma^+$  can be generated by a type-0 grammar. So, there exists a rewrite system  $R$  (resp.  $R'$ ) over  $\Sigma \cup N$ , for some alphabet  $N$  (resp.  $N'$ ) such that there exists  $s$  (resp.  $s'$ ) in  $N$  (resp.  $N'$ ) s.t., for every  $u$  in  $\Sigma^+$ ,  $u \overset{*}{\rightarrow}_R s$  (resp.  $u \overset{*}{\rightarrow}_{R'} s'$ ) iff  $u$  belongs (resp. does not belong) to  $L$ . A type-0 grammar  $G = (N, \Sigma, S, R)$  is a *context-free grammar* when each its rule is of the form  $A \rightarrow w$  where  $A$  is in  $N$ .

### 3 Query Containment and Rewriting

#### 3.1 From word constraints to rewrite rules

As already explained, we can view word constraints and rewrite rules as similar objects. The rewrite rule  $u \rightarrow v$  is naturally mapped to the constraint  $u \sqsubseteq v$  and vice-versa. So given a set of constraints  $\mathcal{C}$  we may consider it as a rewrite system and simply write  $u \overset{*}{\rightarrow}_{\mathcal{C}} v$  to mean that the rewrite system naturally associated with  $\mathcal{C}$  rewrites the word  $u$  to  $v$ . Similarly, given a rewrite system  $R$ , we may write  $D \models R$ ,  $u \sqsubseteq_R^\omega v$  or  $u \sqsubseteq_R^f v$  to denote the fact that in the set of constraints  $\mathcal{C}_R$  that is naturally associated with  $R$ , we have  $D \models \mathcal{C}_R$ ,  $u \sqsubseteq_{\mathcal{C}_R}^\omega v$  or  $u \sqsubseteq_{\mathcal{C}_R}^f v$ . *In the sequel, we will often conflate the set of word constraints and its naturally associated rewrite system.*

#### 3.2 Comparing $\sqsubseteq_R^\omega$ , $\sqsubseteq_R^f$ and $\overset{*}{\rightarrow}_R$

For a set of word constraints  $R$ , a natural question is then how the  $\overset{*}{\rightarrow}_R$ ,  $\sqsubseteq_R^\omega$  and  $\sqsubseteq_R^f$  are related. We are first going to see that  $\overset{*}{\rightarrow}_R$  and  $\sqsubseteq_R^\omega$  coincide, using a construction inspired from [1]:

► **Theorem 2.** *Given a set of word constraints  $R$ , we have*

$$u \sqsubseteq_R^\omega v \text{ iff } u \overset{*}{\rightarrow}_R v .$$

**Proof.** The right to left part of the equivalence does not present any difficulty. For the other direction, let  $D = (V, E)$  the  $\omega$ -database defined by  $V = \Sigma^*$  and  $E = \{(v, a, u) \mid u \overset{*}{\rightarrow}_R va\}$ . An easy induction shows that there is a path labeled  $w$  between the vertices  $v$  and  $u$  iff  $u \overset{*}{\rightarrow}_R vw$ . So if there is a path labeled  $w$  between the vertices  $v$  and  $u$  and  $w \overset{*}{\rightarrow}_R t$ , then  $u \overset{*}{\rightarrow}_R vw \overset{*}{\rightarrow}_R vt$  and so there is a path labeled  $t$  between  $v$  and  $u$ . Thus if  $w \overset{*}{\rightarrow}_R t$ , then  $D \models w \sqsubseteq t$ . This shows that  $D \models R$ . Now, if  $u \sqsubseteq_R^\omega v$ , as there is a path labeled by  $u$  from the vertex  $\varepsilon$  to the vertex  $u$ , we get that there is also a path labeled by  $v$  from  $\varepsilon$  to  $u$ , and then, by what precedes that  $u \overset{*}{\rightarrow}_R v$ . ◀

This theorem tells us also the conditions under which query inclusion holds.

► **Corollary 3.** *Given a set of word constraints  $R$  and two queries  $Q_1$  and  $Q_2$  on that alphabet, the following are equivalent:*

- $Q_1 \sqsubseteq_R^\omega Q_2$ ,
- for every  $p$  in  $Q_1$ ,  $D_R(p) \cap Q_2 \neq \emptyset$ .
- $Q_1 \subseteq A_R(Q_2)$ .

**Proof.** Consider the  $\omega$ -database used in the proof of Theorem 2. If  $Q_1 \sqsubseteq_R^\omega Q_2$ , as  $D \models R$ , for every  $p$  in  $Q_1$ , there is a path labeled by some  $q$  in  $Q_2$  from the vertex  $\epsilon$  to the vertex  $p$ , so by what precedes,  $D_R(p) \cap Q_2 \neq \emptyset$ . The other implications do not present any difficulty. ◀

An important remark is that our characterization fails if we authorize rewrite rules with empty right hand sides, i.e. of the form  $p \rightarrow \epsilon$ . For example, consider the rewrite system  $R$  that contains only one rule  $a \rightarrow \epsilon$ . Clearly we do not have  $a \xrightarrow{*}_R aa$ , however, for every  $\omega$ -database, if there is a path labeled  $a$  between two nodes  $x$  and  $y$ , as  $a \rightarrow_R \epsilon$ , we must have  $x = y$ , so for every  $k$  there is path labeled  $a^k$  from  $x$  to  $y$ : in that case  $a \sqsubseteq_R^\omega aa$  while it is not the case that  $a \xrightarrow{*}_R aa$ .

We have seen in the introduction that contrary to what is stated in of [14, Theorem 2], it is not true that  $\sqsubseteq_R^f$  and  $\xrightarrow{*}_R$  coincide. To summarize, we get:

► **Theorem 4.**

- $u \xrightarrow{*}_R v$  iff  $u \sqsubseteq_R^\omega v$  .
- $Q_1 \sqsubseteq_R^\omega Q_2$  iff  $Q_1 \subseteq A_R(Q_2)$  .
- If  $Q_1 \sqsubseteq_R^\omega Q_2$ , then  $Q_1 \sqsubseteq_R^f Q_2$  .
- In general,  $u \sqsubseteq_R^f v$  does not imply that  $u \sqsubseteq_R^\omega v$  .

### 3.3 Characterizing $\sqsubseteq_R^f$ : from query containment to non-separability

We have seen that  $u \sqsubseteq_R^f v$  and  $u \xrightarrow{*}_R v$  do not coincide. However, we will give a precise characterization of  $Q_1 \sqsubseteq_R^f Q_2$  that uses closure under  $R$ :

► **Theorem 5.** *The following propositions are equivalent:*

- $Q_1 \sqsubseteq_R^f Q_2$ ,
- every regular language closed under  $R$  that intersects with  $Q_1$  intersects with  $Q_2$ .

**Proof.** Suppose that it is not the case that  $Q_1 \sqsubseteq_R^f Q_2$ . Then there exists  $D$ , a model of  $R$  with two vertices  $x$  and  $y$  so that:

- there is a path of  $Q_1$  labeled  $q$  from  $x$  to  $y$ ,
- there is no path labeled by a word of  $Q_2$  from  $x$  to  $y$ .

Seeing  $D$  as an automaton with initial state  $x$  and final state  $y$ , it must be the case that:

- it defines a regular language that is closed under  $R$ ,
- it intersects with  $Q_1$ ,
- it does not intersect with  $Q_2$ .

So there is a regular language closed under  $R$  that intersects with  $Q_1$  and does not intersect with  $Q_2$ .

We now suppose that there is a regular language  $K$  closed under  $R$  that intersects with  $Q_1$  (i.e.  $K \cap Q_1 \neq \emptyset$ ) and does not intersect with  $Q_2$  (i.e.  $K \cap Q_2 = \emptyset$ ). We will build a database  $D$  so that  $D \models R$  and  $D$  does not satisfy  $Q_1 \sqsubseteq_R^f Q_2$ .

Let  $K_1, \dots, K_n$  be the finite set of *left residuals* of  $K$ . The left residual of  $K$  by a word  $q$ , noted  $q^{-1}K$ , is the language  $q^{-1}K = \{p \mid qp \in K\}$ . A language is a left residual of  $K$  when it is of the form  $q^{-1}K$  for some  $q$ . It is well-known that a language is regular iff the set of its left residuals is finite. We start by making the following remark about the  $K_i$ 's:

► **Lemma 6.** *For every  $i$  in  $[n]$ ,  $K_i$  is closed under  $R$ .*

**Proof.** Take  $i$  in  $[n]$ ,  $p$  in  $K_i$  and  $p'$  such that  $p \xrightarrow{*}_R p'$ . There is  $q$  so that  $K_i = q^{-1}K$  and  $qp$  is in  $K$ . Since  $K$  is closed under  $R$ ,  $qp'$  is also in  $K$  implying that  $p'$  is in  $q^{-1}K = K_i$ . ◀

We define the database  $D$  as follows:

- the set of vertices is  $\{K_1, \dots, K_n\}$ ,
- there is an edge labeled  $a$  between  $K_i$  and  $K_j$  iff  $K_j \subseteq a^{-1}K_i$ .

► **Lemma 7.** *There is a path in  $D$  labeled by  $p \in \Sigma^+$  between  $K_i$  and  $K_j$  iff  $p^{-1}K_i \supseteq K_j$ .*

**Proof.** We proceed by induction on  $p$ .

When  $p = a$ , the conclusion directly follows from the definition.

Let  $p = p'a$  with  $p' \in \Sigma^+$ . We first suppose that there is a path labeled  $p$  from  $K_i$  to  $K_j$ . So, there is  $K_k$  so that there is a path labeled  $p'$  from  $K_i$  to  $K_k$  and there is an arc labeled  $a$  between  $K_k$  and  $K_j$ . From the induction hypothesis, we have that  $p'^{-1}K_i \supseteq K_k$  and by definition  $a^{-1}K_k \supseteq K_j$ , thus  $p^{-1}K_i = a^{-1}p'^{-1}K_i \supseteq a^{-1}K_k \supseteq K_j$ . Suppose now that  $p^{-1}K_i \supseteq K_j$ , we let  $K_k = p'^{-1}K_i$ . By induction, there is a path labeled  $p'$  between  $K_i$  and  $K_k$ , and moreover  $a^{-1}K_k = p^{-1}K_i \supseteq K_j$  so that there is an edge labeled  $a$  between  $K_k$  and  $K_j$ . Therefore there is a path labeled  $p$  between  $K_i$  and  $K_j$ . ◀

► **Lemma 8.** *If there is a path labeled  $p$  between  $K_i$  and  $K_j$ , for any constraint  $p \sqsubseteq q$  in  $R$ , there is a path labeled  $q$  between  $K_i$  and  $K_j$ .*

**Proof.** If there is a path labeled  $p$  between  $K_i$  and  $K_j$ , then  $p^{-1}K_i \supseteq K_j$  from Lemma 7. So, if  $t$  belongs to  $K_j$ ,  $pt$  belongs to  $K_i$ . As  $p \sqsubseteq q$  in  $R$ , we have  $p \rightarrow_R q$  and therefore  $pt \rightarrow_R qt$ . Now, from Lemma 6,  $qt$  also belongs to  $K_i$  and thus  $t$  is in  $q^{-1}K_i$ . Consequently  $q^{-1}K_i \supseteq K_j$  and Lemma 7 implies that there is a path labeled  $q$  between  $K_i$  and  $K_j$ . ◀

The previous lemma shows that  $D \models R$ . Now let  $x = K$  and  $y = q^{-1}K$  with  $q \in K \cap Q_1$  (recall that  $K \cap Q_1 \neq \emptyset$ ).

We now show that the set of words that label paths between  $x$  and  $y$  intersects with  $Q_1$  and is included in  $K$  and so does not intersect with  $Q_2$ . It intersects with  $Q_1$  as from Lemma 7, there is a path labeled by  $q$  between  $K$  and  $q^{-1}K$ . When there is a path labeled by  $p$  between  $K$  and  $q^{-1}K$ , we have that  $p^{-1}K \supseteq q^{-1}K$  (Lemma 7). Now since  $q \in K$ , we have that  $\varepsilon \in q^{-1}K$  and therefore  $\varepsilon$  is also an element of  $p^{-1}K$  so  $p$  belongs to  $K$  and does not belong to  $Q_2$  by hypothesis.

In a nutshell, we have  $D \models R$ , there is a path between  $x$  and  $y$  labeled by a word of  $Q_1$  (the word  $q$ ) and no path labeled by a word in  $Q_2$ . This finally shows that it is not the case that  $Q_1 \sqsubseteq_R^f Q_2$ . ◀

So, we get as corollaries:

► **Corollary 9.** *The following propositions are equivalent:*

- $p \sqsubseteq_R^f Q_2$ ,
- every regular language closed under  $R$  that contains  $p$  intersects with  $Q_2$ .

► **Corollary 10.** *The following propositions are equivalent:*

- $p_1 \sqsubseteq_R^f p_2$ ,
- every regular language closed under  $R$  that contains  $p_1$  contains  $p_2$ .

## 4 About (non) finite controllability

### 4.1 Non finitely controllable systems

We now show how to construct examples where we have  $u \sqsubseteq_R^f v$  and we do not have  $u \sqsubseteq_R^\omega v$ . We first illustrate the idea of the construction by taking  $R = \{c \rightarrow acb\}$ . The set of descendants of  $c$  by  $R$  is  $\{a^n cb^n \mid n \in \mathbb{N}\}$ . Every finite database  $D$  such that  $D \models R$  and that contains a path labeled  $c$  between two nodes has necessarily (by a usual pumping argument) a path labeled  $a^n cb^m$  with  $n > m$  between these two nodes. Now, let  $T = \{acb \rightarrow c, ac \rightarrow c, ac \rightarrow 0\}$ . It is easy to see that any word  $a^n cb^m \xrightarrow{*}_T 0$  iff  $n > m$ . Thus, in every finite database  $D$  such that  $D \models R \cup T$ , if there is a path labeled by  $c$  between two vertices, then there is also a path labeled by  $0$ :  $c \sqsubseteq_{R \cup T}^f 0$ . However, one can check that  $c$  does not rewrite to  $0$  with  $R \cup T$ . The idea underlying this construction can be used for any rewrite system  $R$  and any word  $u$  such that  $D_R(u)$  is recursive but not regular.

► **Proposition 11.** *Let any set of word constraints  $R$  and any word  $u$  such that  $D_R(u)$  is recursive but not regular. Let  $R' = R \cup T \cup B$  defined as above: then  $u \sqsubseteq_{R'}^f 0$  while it is not the case that  $u \sqsubseteq_{R'}^\omega 0$ .*

**Proof.** We assume that the symbols used by  $R$  are taken from the finite set  $\Sigma$ . We now take a finite set  $\bar{\Sigma}$  with the same number of elements as  $\Sigma$  and a bijection between  $\Sigma$  and  $\bar{\Sigma}$ . We define  $B = \{a \rightarrow \bar{a} \mid a \in \Sigma\}$ . Given a word  $w$  in  $(\Sigma \cup \Gamma)^*$  with  $\Gamma \cap \Sigma = \emptyset$ , we write  $\bar{w}$ , for the word obtained by replacing all occurrences of  $a$  in  $\Sigma$  by  $\bar{a}$  and leaving other letters unchanged. As we suppose that  $D_R(u)$  is recursive, we can assume the existence of a rewrite system  $T$  based on an alphabet  $\Gamma$  disjoint from  $\Sigma$  and that contains  $\bar{\Sigma}$  and  $\{0\}$  so that:  $\bar{v} \rightarrow_T 0$  iff  $v$  is not in  $D_R(u)$ . Furthermore, we can suppose that  $0$  does not occur in  $lhs(T)$ . We take  $R' = R \cup T \cup B$ .

► **Lemma 12.** *For every  $v$  in  $\Sigma^+$ ,  $v \xrightarrow{*}_{R'} 0$  iff there is  $w$  in  $\Sigma^+$  so that  $v \xrightarrow{*}_R w$  and  $w \notin D_R(u)$ .*

**Proof.** The *if* part of the statement is a simple consequence of the definitions. For the *only if* part we prove a slightly stronger property. Given a word  $v$  we prove that there is  $w$  so that:

- $v \xrightarrow{*}_R w$  and
- $\bar{w} \xrightarrow{*}_T 0$ .

Indeed, by Lemma 1, using the fact that the alphabet of  $rhs(T)$  is disjoint from the alphabet of  $lhs(R \cup B)$ , that the alphabet of  $rhs(B)$  is disjoint from the alphabet of the  $lhs(R)$ , we get that there exists  $w$  in  $\Sigma^+$ ,  $w'$  in  $(\Sigma \cup \bar{\Sigma})^+$  such that  $v \xrightarrow{*}_R w \xrightarrow{*}_B w' \xrightarrow{*}_T 0$ . As  $0$  is only produced by  $T$  and as the alphabet of  $lhs(T)$  is disjoint from  $\Sigma$ , we have  $w' = \bar{w}$ . ◀

Now, let  $K$  be a language containing  $u$  that is regular and closed under  $R'$ . Then  $K \cap \Sigma^*$  is regular and contains  $D_R(u)$ : as  $D_R(u)$  is not regular  $K \cap \Sigma^*$  contains a word  $v$  in  $\Sigma^*$  that is not in  $D_R(u)$ : then  $v \xrightarrow{*}_B \bar{v} \xrightarrow{*}_T 0$ : as  $K$  is closed under  $R'$ ,  $K$  contains  $0$  and by Theorem 5,  $u \sqsubseteq_{R'}^f 0$  while it is not the case that  $u \xrightarrow{*}_{R'} 0$ . ◀

### 4.2 Finite controllability: word queries vs RPQs

A consequence of [14, Theorem 2] that is also false, is that  $Q_1 \sqsubseteq_R^f Q_2$  iff for every  $q_1$  in  $Q_1$  there is  $q_2$  in  $Q_2$  so that  $q_1 \sqsubseteq_R^f q_2$  [14, Lemma 3]. We construct here an example of a rewrite system for which this property does not hold. We take the following rewrite system  $R$ :



$$\begin{array}{ll}
a \rightarrow aab & aab \rightarrow a \\
b \rightarrow abb & abb \rightarrow b \\
ab \rightarrow ba & ba \rightarrow ab
\end{array}$$

Let  $p, q$  be two words. If  $p = \epsilon$  or  $q = \epsilon$ , it is easy to check that  $p \xrightarrow{*}_R q$  iff  $p = q$  iff  $p \sqsubseteq_R^f q$ . Let us now suppose that both are non empty. First, it is easy to check that  $p \xrightarrow{*}_R q$  iff  $|q|_a - |q|_b = |p|_a - |p|_b$ <sup>1</sup>. Second,  $p \sqsubseteq_R^f q$  iff  $p \xrightarrow{*}_R q$ . Indeed, let  $N_q = |q|_a - |q|_b$ ,  $N_p = |p|_a - |p|_b$ . If we do not have  $p \xrightarrow{*}_R q$ , then  $N_q \neq N_p$ . Let  $K = \{w \mid |w|_a - |w|_b \equiv N_p \pmod{(N_q+1)(N_p+1)}\}$ . The language  $K$  is regular and closed under  $R$ , contains  $p$  and does not contain  $q$ . So, using Theorem 5, we do not have  $p \sqsubseteq_R^f q$ .

Now, let  $p = ab$  and let  $Q$  be the regular language  $b^+$ . By what precedes, there is no  $q$  in  $Q$  so that  $p \sqsubseteq_R^f q$ . Now, let  $K$  a regular language that contains  $p$  and that is closed under  $R$ . Then  $K$  contains  $\{a^n b^n \mid n > 0\}$ . Using the pumping lemma for regular languages we deduce that for some  $m > 0$  we must have that  $a^n b^{n+m}$  is in  $K$ . As  $K$  is closed under  $R$ , it contains  $b^m$ .

So, by Theorem 5, we do have  $p \sqsubseteq_R^f Q$ , while there is no word  $q$  of  $Q$  so that  $p \sqsubseteq_R^f q$ .

Furthermore, by Corollary 3,  $P \sqsubseteq_R^\omega Q$  iff for every  $q_1$  in  $P$  there is  $q_2$  in  $Q_2$  so that  $q_1 \sqsubseteq_R^\omega q_2$ . So,  $R$  provides an example of system such that containment of word queries is finitely controllable whereas the containment for regular path queries is not:

► **Theorem 13.** *There are sets of word constraints for which the containment of word queries is finitely controllable, while the containment of regular path queries is not.*

## 5 Query containment under word constraints is undecidable

In [14], the proof of undecidability of query containment under word constraints is a mere corollary of the assertion that  $u \sqsubseteq_R^f v$  and  $u \xrightarrow{*}_R v$  coincide. As we have seen earlier, this assertion is false. However, query containment under word constraints is actually undecidable. In this section, we give a proof of that fact. Notice that this result can also be derived from [9, Theorem 15].

► **Theorem 14.** *wwQC is undecidable.*

**Proof.** The undecidability result is obtained by reduction from the problem of the separability of context-free languages by some regular language. Formally this decision problem is stated as follows:

*Input* two context free grammars  $G_1$  and  $G_2$

*Question* Is there a regular language  $R$  so that  $L(G_1) \subseteq R$  and  $L(G_2) \cap R = \emptyset$  ?

This problem is known to be undecidable [15].

Take  $G_1 = (N_1, \Sigma, S_1, R_1)$  and  $G_2 = (N_2, \Sigma, S_2, R_2)$  two context free grammars on the alphabet  $\Sigma$ . We assume w.l.o.g. that

- $N_1 \cap N_2 = \emptyset$ ,
- they do not have rules of the form  $A \rightarrow \epsilon$ ,
- they are reduced (every non-terminal is reachable from the start symbol and defines a non-empty language).

We let  $R$  be the rewrite system on the alphabet  $\Gamma = \Sigma \uplus N_1 \uplus N_2$  containing the rules of  $R_1$  and  $R_2^{-1}$ .

<sup>1</sup>  $|u|_x$  denotes the number of occurrences of the letter  $x$  in the word  $u$ .

► **Lemma 15.** *Given  $u$  and  $v$  in  $\Gamma^*$ , we have that  $u \xrightarrow{*}_R v$  iff there is  $w$  so that  $u \xrightarrow{*}_{R_1} w$  and  $w \xrightarrow{*}_{R_2^{-1}} v$ . Furthermore, if  $u$  does not contain any non-terminal of  $G_2$  and  $v$  does not contain any non-terminal of  $G_1$ , then  $w$  is in  $\Sigma^+$ .*

**Proof.** As we have supposed that  $N_1 \cap N_2 = \emptyset$ , it must be the case that left hand side of a rule in  $R_1$  does not share any symbol with the right hand side of a rule in  $R_2^{-1}$ . So, we get the first part of the lemma by Lemma 1. Finally, as  $R_1$  (resp.  $R_2^{-1}$ ) cannot generate (resp. eliminate) nonterminals of  $G_2$  (resp.  $G_1$ ),  $w$  does not contain any non-terminal symbol of  $G_2$  (resp.  $G_1$ ). ◀

We are going to show that it is not the case that  $S_1 \sqsubseteq_R^f S_2$  iff there is a regular language that separates  $L(G_1)$  and  $L(G_2)$ .

Suppose that it is not the case that  $S_1 \sqsubseteq_R^f S_2$ : by what precedes, there exists a regular language  $L$  closed under  $R$  that contains  $S_1$  and does not contain  $S_2$ . As it is closed under  $R$ , it is closed under  $R_1$ : as it contains  $S_1$ , it contains  $L(G_1)$ . Similarly, it is closed under  $R_2^{-1}$  and as it does not contain  $S_2$ , so it does not contain any word of  $L(G_2)$ :  $L$  is a regular language that separates  $L(G_1)$  and  $L(G_2)$ .

Suppose now that  $L$  is a regular language that separates the languages of  $G_1$  and of  $G_2$ , e.g. that contains  $L(G_1)$  and does not intersect with  $L(G_2)$ . We can suppose  $L \subseteq \Sigma^*$ . Let us note  $\Sigma_1$  the alphabet  $\Sigma$  enriched with the non terminals of  $G_1$ . By Theorem 5, we only have to prove that there is a regular language closed under  $R$  that contains  $S_1$  and does not contain  $S_2$ .

Let us first notice that  $R_1$  (resp.  $R_2^{-1}$ ) preserves regularity by ascendants (resp. descendants).

Let  $K_1 = \Sigma_1^*/A_{R_1}(\Sigma^*/L)$ .  $K_1$  is regular. Let us prove that  $K_1$  is closed under  $R_1$ ; let  $u$  in  $K_1$ ,  $u \xrightarrow{*}_{R_1} v$ : then  $v$  belongs to  $\Sigma_1^*$ ; if  $v$  does not belong to  $K_1$ ,  $v$  belongs to  $A_{R_1}(\Sigma^*/L)$  and, as  $u \xrightarrow{*}_{R_1} v$ ,  $u$  belongs to  $A_{R_1}(\Sigma^*/L)$ , which contradicts the fact that  $u$  belongs to  $K_1$ . By construction,  $K_1 \cap \Sigma^* \subseteq L$ . As  $D_{R_1}(S_1) \cap \Sigma^* = L(G_1)$ ,  $D_{R_1}(S_1) \cap \Sigma^* \subseteq L$ :  $S_1$  belongs to  $K_1$  and as  $K_1$  is closed under  $R_1$ ,  $K_1 \cap \Sigma^* \supseteq L$ . So,  $K_1 \cap \Sigma^* = L$ .

Now, let  $K = D_{R_2^{-1}}(K_1)$ ;  $S_1$  belongs to  $K$ , as  $S_1$  belongs to  $K_1$ . By Lemma 15,  $K$  is closed by  $R$ , as  $K_1$  is closed by  $R_1$ . Furthermore, by the same lemma,  $K \cap \Sigma_2^* = D_{R_2^{-1}}(K_1 \cap \Sigma^*)$ , i.e.  $K \cap \Sigma_2^* = D_{R_2^{-1}}(L)$ . As  $L \cap L(G_2)$  is empty,  $K$  does not contain  $S_2$ .

So,  $K$  is a regular language closed under  $R$  that contains  $S_1$  and does not contain  $S_2$ : by Theorem 5, it is not the case that  $S_1 \sqsubseteq_R^f S_2$ . So  $L(G_1)$  and  $L(G_2)$  are separable by a regular language iff we do not have  $S_1 \sqsubseteq_R^f S_2$ .

So, given  $R, u, v$ , it is undecidable whether  $u \sqsubseteq_R^f v$ . ◀

## 6 How to ensure decidability and finite controllability of $QC$ ?

As we have proven that generally,  $u \sqsubseteq_R^f v$  does not necessarily imply that  $u \xrightarrow{*}_R v$ , we naturally wonder which classes of rewrite systems ensure the equivalence between  $u \sqsubseteq_R^f v$  and  $u \xrightarrow{*}_R v$ . We will see later that this equivalence is in general undecidable for arbitrary rewrite systems. This entails, that we must content ourselves with finding particular restrictions which have this property without ever having a complete and effective characterization. More generally, we can also try to identify classes of rewrite systems which ensure the equivalence between  $Q_1 \sqsubseteq_R^f Q_2$  and  $Q_1 \sqsubseteq_R^\omega Q_2$  for any RPQs  $Q_1$  and  $Q_2$ .

By Subsection 3.3, we get easily (Proofs in Appendix B) the three following lemmas:

► **Lemma 16.** *For any word query  $p$  and RPQ  $Q$ , if  $D_R(p)$  is regular, then  $p \sqsubseteq_R^f Q$  iff  $p \sqsubseteq_R^\omega Q$ .*

► **Lemma 17.** *There is a set of word constraints  $R$  and RPQs  $P$  and  $Q$  so that  $D_R(P)$  is regular,  $P \sqsubseteq_R^f Q$ , but we do not have  $P \sqsubseteq_R^\omega Q$ .*

► **Lemma 18.** *For any RPQs  $Q_1, Q_2$ , if  $A_R(Q_2)$  is regular, then  $Q_1 \sqsubseteq_R^f Q_2$  iff  $Q_1 \sqsubseteq_R^\omega Q_2$ .*

So, an obvious property ensuring finite controllability of a rewrite system  $R$  is the *preservation of regularity* by  $R$  or its inverse  $R^{-1}$ . A rewrite system  $R$  (resp.  $R^{-1}$ ) *preserves regularity* when for every regular language  $Q$ ,  $D_R(Q)$  (resp.  $D_{R^{-1}}(Q) = A_R(Q)$ ) is a regular language. Lemma 17 tells us that it is not enough that the rewrite system preserves the regularity of a particular regular language.

► **Corollary 19.** *Let  $R$  a rewrite system. If  $R$  (resp.  $R^{-1}$ ) preserves regularity, query containment is finitely controllable.*

**Proof.** If  $R^{-1}$  preserves regularity, we get the result by Lemma 18. If  $R$  preserves regularity, let us suppose that  $Q_1 \sqsubseteq_R^f Q_2$ . It implies that for every  $p$  in  $Q_1$ ,  $p \sqsubseteq_R^f Q_2$ , and then by Lemma 16  $p \sqsubseteq_R^\omega Q_2$ : so,  $Q_1 \sqsubseteq_R^\omega Q_2$ . ◀

We say that  $R$  *effectively preserves regularity* when it preserves regularity and when given a regular language  $Q$  (effectively presented by a regular expression or a finite state automaton) it is possible to compute (a representation of)  $D_R(Q)$ . We write  $\text{REWREC}$  to denote this class of rewrite systems and  $\text{REWREC}^{-1}$  the class rewrite systems  $R$  whose inverse  $R^{-1}$  effectively preserves regularity.

► **Corollary 20.**

- *The problem  $\text{WUQC}(\text{REWREC})$  is decidable.*
- *The problem  $\text{UUQC}(\text{REWREC}^{-1})$  is decidable.*

Deciding whether a rewrite system preserves regularity is an undecidable property [17]. However, several classes of string rewrite systems that effectively preserve regularity have been identified, e.g. monadic systems [6] and match-bounded systems [11]. Monadic systems are systems whose rules have a letter as right-hand side – so the corresponding TGDs can be viewed as a Datalog program. A context-free grammar can be viewed as the inverse of a monadic rewrite system and so query containment is finitely controllable and decidable for the corresponding class of word constraints.

The first part of Corollary 20 cannot be generalized to  $\text{UUQC}(\text{REWREC})$  which happens to be undecidable. We prove a slightly stronger result in Theorem 21 using the problem of universality of context-free languages which is well known to be undecidable. It can be stated as follows:

*Input* A context free grammar  $G$   
*Question*  $\mathcal{L}(G) = \Sigma^+$ ?

The reduction is as follows: take a context free grammar  $G = (N, \Sigma, S, \Delta)$  (w.l.o.g. we suppose that  $\epsilon$  does not belong to  $G$  and does not occur in r.h.s. of  $\Delta$ ). The rewrite system  $R = \Delta^{-1}$  is monadic and thus effectively preserves regularity [6]. Thus, Lemma 18 and Theorem 2 tell us that  $\Sigma^+ \sqsubseteq_{R^{-1}}^f S$  is equivalent to the universality problem for  $G$  and is thus undecidable. This gives us an alternative proof of [14, Theorem 4]. Both proofs rely on similar constructions but the proof in [14] relies on a false theorem, namely [14, Theorem 3].

► **Theorem 21.** *The problem  $\text{UWQC}(\text{REWREC})$  is undecidable.*

## 17:14 Containment of Regular Path Queries Under Path Constraints

Our characterization leaves room for finding rewrite systems that do not preserve regularity but for which query containment is finitely controllable. The following proposition (Proof in Appendix C) states a modularity property: if two sets of word constraints are alphabet-disjoint and if both ensure finite controllability of query containment, their union does also ensure finite controllability.

► **Proposition 22.** *Let  $R_1$  and  $R_2$  two rewrite systems on disjoint alphabets  $\Sigma_1$  and  $\Sigma_2$  that ensure finite controllability of regular query containment. Then,  $R = R_1 \cup R_2$  also ensures finite controllability of regular query containment.*

As the preservation of regularity of rewrite systems is not in general closed under union, this proposition allows us to construct rewrite systems  $R$  that ensure finite controllability of regular queries containment while neither  $R$  nor  $R^{-1}$  preserves regularity. E.g., let  $R_1 = \{c \rightarrow acb\}$  and  $R_2 = \{dfe \rightarrow f\}$ .  $R_1^{-1}$  and  $R_2$  are monadic and so preserve regularity and then  $R_1$  and  $R_2$  ensure both finite controllability. By the preceding proposition,  $R = \{c \rightarrow acb, dfe \rightarrow f\}$  ensures finite controllability. However, neither  $R$  nor  $R^{-1}$  preserve regularity, as  $D_R(c) = \{a^n cb^n \mid n \in \mathcal{N}\}$  and  $A_R(f) = \{d^n fe^n \mid n \in \mathcal{N}\}$ .

### 7 Finite controllability is undecidable

We are now looking at the decidability of *finite controllability* (wwFC) and of *uniform finite controllability* (wwUFC) defined in Section 1.

As  $u \sqsubseteq_R^\omega v$  implies  $u \sqsubseteq_R^f v$ , we focus on the undecidability of  $u \sqsubseteq_R^\omega v$  under the hypothesis that  $u \sqsubseteq_R^f v$ . We use a reduction from the following undecidable problem (Proof in Appendix D):

► **Lemma 23.** *The following problem is undecidable:*

*Input*  $L_1, L_2$  two recursive sets that are not separable by a regular set.  
*Question* Is  $L_1 \cap L_2$  empty?

So, let  $L_1, L_2$  be two recursive sets on the alphabet  $\Sigma^+$  that are not separable by a regular set. As previously, we take a finite set  $\bar{\Sigma}$  in bijection with  $\Sigma$ . We define  $B = \{a \rightarrow \bar{a} \mid a \in \Sigma\}$ . Given an alphabet  $\Gamma$  disjoint from  $\Sigma$  and a word  $w$  in  $(\Sigma \uplus \Gamma)^*$ , we write  $\bar{w}$ , for the word obtained by replacing all occurrences of  $a$  in  $\Sigma$  by  $\bar{a}$  and leaving other letters unchanged.

As  $L_1, L_2$  are recursive, there exist two rewrite systems :

- $R_1$  on an alphabet  $\Sigma_1$  containing  $\Sigma$  and a symbol  $s_1$  such that for any  $u$  in  $\Sigma^*$ ,  $s_1 \xrightarrow{*}_{R_1} u$  iff  $u \in L_1$ .
- $R_2$  on an alphabet  $\Sigma_2$  containing  $\bar{\Sigma}$  and a symbol  $s_2$  such that for any  $u$  in  $\Sigma^*$ ,  $\bar{u} \xrightarrow{*}_{R_2} s_2$  iff  $u \in L_2$ .

We can suppose that  $\Sigma_1 \cap \Sigma_2 = \emptyset$ . Let  $\Delta = \Sigma_1 \cup \Sigma_2 \cup \{\#_l, \#_r, g\}$  where  $\#_l, \#_r$  and  $g$  are fresh symbols. We define:

$$\begin{aligned} R_{\#} &= \{x \rightarrow \#_l s_1 \#_r \mid x \in \Delta \setminus \{\#_l, \#_r\}\} \\ R_g &= \{\#_l s_2 \#_r \rightarrow g\} \cup \{g \rightarrow xg, g \rightarrow x, xg \rightarrow g, gx \rightarrow g \mid x \in \Delta\} \\ R_{L_1, L_2} &= R_1 \cup R_2 \cup R_{\#} \cup B \cup R_g. \end{aligned}$$

In the sequel, we denote  $R_{L_1, L_2}$  by  $R$ . Then, we get (Proof in Appendix E):

► **Lemma 24.**

1. If  $L_1 \cap L_2 \neq \emptyset$ ,  $u \xrightarrow{*}_R v$  for any  $(u, v)$  in  $\Delta^+ \setminus \{\#_l, \#_r\}^* \times \Delta^+$ .
2. If  $L_1 \cap L_2 = \emptyset$ , we do not have  $\#_l s_1 \#_r \xrightarrow{*}_R \#_l s_2 \#_r$ .
3.  $\#_l s_1 \#_r \sqsubseteq_R^f \#_r s_2 \#_r$ .
4. If  $u \in \{\#_l, \#_r\}^*$ , then  $u \sqsubseteq_R^f v$  (resp.  $u \xrightarrow{*}_R v$ ) iff  $u = v$ .
5.  $u \sqsubseteq_R^f v$  iff  $u \in \Delta^+ \setminus \{\#_l, \#_r\}^*$  or  $u = v$ .

A consequence of Lemma 24 is that deciding equivalence of  $\sharp_l s_1 \sharp_r \sqsubseteq_R^f \sharp_r s_2 \sharp_r$  and  $\sharp_l s_1 \sharp_r \xrightarrow{*}_R \sharp_r s_2 \sharp_r$  amounts to decide non emptiness of  $L_1 \cap L_2$ . More generally, for every  $u, v \in \Delta^+$ ,  $u \sqsubseteq_R^f v$  is equivalent to  $u \xrightarrow{*}_R v$  only when  $L_1 \cap L_2 \neq \emptyset$ . Therefore we get the undecidability of finite controllability and uniform finite controllability of a rewrite system:

► **Theorem 25.** *wwFC and wwUFC are undecidable.*

Furthermore, let  $C$  be the class of systems  $R_{L_1, L_2}$  for  $L_1, L_2$  recursive sets on the alphabet  $\Sigma$  that are not separable by a regular set. On the one hand, for any rewrite system  $R$  in  $C$ ,  $u \sqsubseteq_R^f v$  iff  $u \in \Delta^+ / \{\sharp_l, \sharp_r\}^*$  or  $u = v$ . So, for any rewrite system  $R$  in  $C$ ,  $\sqsubseteq_R^f$  is decidable. On the other hand,  $s_1 \xrightarrow{*}_R s_2$  is equivalent to  $L_1 \cap L_2 = \emptyset$  so is undecidable in  $C$ . So, we get:

► **Proposition 26.** *There exists a class of rewrite systems for which the wwQC is decidable whereas  $\text{wwQC}^\omega$  is undecidable.*

This proposition contradicts [14, Corollary 2] that is a consequence of [14, Theorem 2].

## 8 Conclusion

Starting with an error in the proof of [14], we have studied the containment of RPQs under word constraints. Contrary to what was claimed in [14], we have showed that this property is not finitely controllable in general. We also have given counter-examples to properties that were corollaries of this false claim and alternate proofs to those that were correct.

For this, we have studied the relation between word constraints and rewrite systems. We have given a precise characterization of  $P \sqsubseteq_R^f Q$  in terms of separability and closure under rewriting by  $R$ . This characterization has played a key role in identifying the properties of query containment in this setting and in giving a correct proof of the undecidability of the containment problem.

The stage being set we have studied further properties of finite controllability in this setting. In particular, we have showed that it is undecidable and we have exhibited some classes of constraints that ensure the finite controllability and the decidability of query containment. This study allowed us to show that the finite controllability of the containment of word queries and that of RPQs do not coincide. More specifically we give examples of constraints for which the containment of word queries is finitely controllable, whereas it is not the case for general RPQs. Interestingly we have also showed that when  $p \sqsubseteq_R^f Q$ , we do not necessarily have  $p \sqsubseteq_R^f q$  for some word  $q$  in  $Q$ , i.e. the “witness” of containment depends on the model.

We observe that for obtaining finite controllability in this setting, it suffices to consider constraints for which the underlying rewrite system preserves regularity by inverse rewriting. We also observe that those for which the underlying rewrite system preserves regularity have nice properties. Such rewrite systems have been widely studied. We show that other rewrite systems can also have interesting properties with respect to that containment problem (as done in Proposition 22).

Finally many of the results of the paper could be extended to RPQ constraints of the form  $P \subseteq u$ , where  $P$  is an RPQ,  $u$  a word. In particular, we think that the characterization of  $u \sqsubseteq_R^f v$  in terms of separability could likely be extended. An interesting consequence would then be that decidability results about finite controllability and query containment would then hold for some classes of RPQs.

---

**References**

---

- 1 Serge Abiteboul and Victor Vianu. Regular path queries with constraints. *Journal of Computer and System Sciences*, 58(3):428–452, 1999. doi:10.1006/jcss.1999.1627.
- 2 Giovanni Amendola, Nicola Leone, and Marco Manna. Finite controllability of conjunctive query answering with existential rules: Two steps forward. In Jérôme Lang, editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 5189–5193. ijcai.org, 2018. doi:10.24963/ijcai.2018/719.
- 3 Yves André, Anne-Cécile Caron, Denis Debarbieux, Yves Roos, and Sophie Tison. Path constraints in semistructured data. *Theoretical Computer Science*, 385(1-3):11–33, oct 2007. doi:10.1016/j.tcs.2007.05.010.
- 4 Pablo Barceló and Gaëlle Fontaine. On the data complexity of consistent query answering over graph databases. *Journal of Computer and System Sciences*, 88:164–194, 2017. doi:10.1016/j.jcss.2017.03.015.
- 5 Meghyn Bienvenu, Pierre Bourhis, Marie-Laure Mugnier, Sophie Tison, and Federico Ulliana. Ontology-mediated query answering for key-value stores. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 844–851. ijcai.org, 2017. doi:10.24963/ijcai.2017/117.
- 6 Ronald V. Book and Friedrich Otto. *String-Rewriting Systems*. Texts and Monographs in Computer Science. Springer, 1993. doi:10.1007/978-1-4613-9771-7.
- 7 R. Büchi. *Regular canonical systems*. Arch. Math. Logik Grundlag., 1964.
- 8 Peter Buneman, Wenfei Fan, and Scott Weinstein. Path constraints in semistructured databases. *Journal of Computer and System Sciences*, 61(2):146–193, 2000. doi:10.1006/jcss.2000.1710.
- 9 Diego Calvanese, Magdalena Ortiz, and Mantas Simkus. Verification of evolving graph-structured data under expressive path constraints. In Wim Martens and Thomas Zeume, editors, *19th International Conference on Database Theory, ICDT 2016, Bordeaux, France, March 15-18, 2016*, volume 48 of *LIPICs*, pages 15:1–15:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICs.ICDT.2016.15.
- 10 Diego Figueira, Santiago Figueira, and Edwin Pin Baque. Finite Controllability for Ontology-Mediated Query Answering of CRPQ. In *International Conference on Principles of Knowledge Representation and Reasoning (KR)*, Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning (KR’20), Rhodes, Greece, sep 2020. doi:10.24963/kr.2020/39.
- 11 Alfons Geser, Dieter Hofbauer, and Johannes Waldmann. Match-bounded string rewriting systems. *Appl. Algebra Eng. Commun. Comput.*, 15(3-4):149–171, 2004. doi:10.1007/s00200-004-0162-8.
- 12 Tomasz Gogacz and Jerzy Marcinkowski. Converging to the chase – A tool for finite controllability. *Journal of Computer and System Sciences*, 83(1):180–206, 2017. doi:10.1016/j.jcss.2016.08.001.
- 13 Georg Gottlob, Marco Manna, and Andreas Pieris. Finite model reasoning in hybrid classes of existential rules. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 1831–1837. International Joint Conferences on Artificial Intelligence Organization, jul 2018. doi:10.24963/ijcai.2018/253.
- 14 Gösta Grahne and Alex Thomo. Query containment and rewriting using views for regular path queries under constraints. In *Proceedings of the Twenty-Second ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 9-12, 2003, San Diego, CA, USA*, pages 111–122, 2003. doi:10.1145/773153.773165.
- 15 Harry B. Hunt III. On the decidability of grammar problems. *J. ACM*, 29(2):429–447, 1982. doi:10.1145/322307.322317.



- 16 D.S. Johnson and A. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. *Journal of Computer and System Sciences*, 28(1):167–189, 1984. doi:10.1016/0022-0000(84)90081-3.
- 17 Friedrich Otto. Some undecidability results concerning the property of preserving regularity. *Theor. Comput. Sci.*, 207(1):43–72, 1998. doi:10.1016/S0304-3975(98)00055-3.
- 18 Riccardo Rosati. On the decidability and finite controllability of query processing in databases with incomplete information. In Stijn Vansummeren, editor, *Proceedings of the Twenty-Fifth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 26-28, 2006, Chicago, Illinois, USA*, pages 356–365. ACM, 2006. doi:10.1145/1142351.1142404.
- 19 Riccardo Rosati. On the finite controllability of conjunctive query answering in databases under open-world assumption. *J. Comput. Syst. Sci.*, 77(3):572–594, 2011. doi:10.1016/j.jcss.2010.04.011.

## A

 Proof of Lemma 1

**Proof.** First let us note that if  $u \rightarrow_{R_2} w \rightarrow_{R_1} v$ , there exists  $w'$  such that  $u \rightarrow_{R_1} w' \rightarrow_{R_2} v$ . Indeed, as  $u \rightarrow_{R_2} w$ ,  $u = u_1 l_2 u_2$ ,  $w = u_1 r_2 u_2$  for some rule  $(l_2, r_2)$  in  $R_2$ . As  $w \rightarrow_{R_1} v$ ,  $w = v_1 l_1 v_2$  for some rule  $(l_1, r_1)$  in  $R_2$  and  $v = v_1 r_1 v_2$ . By hypothesis the letters of  $l_1$  do not occur in  $r_2$ : so either  $v_1 l_1$  is a prefix of  $u_1$ , or  $l_1 v_2$  is a suffix of  $u_2$ . W.l.o.g., let us suppose that  $v_1 l_1$  is a prefix of  $u_1$ :  $w = v_1 l_1 w_1 r_2 u_2$ ,  $v = v_1 r_1 w_1 r_2 u_2$  and  $u = v_1 l_1 w_1 l_2 u_2$ :  $u \rightarrow_{R_1} v_1 r_1 w_1 l_2 u_2 \rightarrow_{R_2} v_1 r_1 w_1 r_2 u_2 = v$ .

Now, let  $u \xrightarrow{*}_R v$ : there exists a derivation  $u \rightarrow_R u_1 \rightarrow_R \dots u_n = v$ . At step  $i$  either a rule of  $R_1$  or a rule of  $R_2$  is applied. An inversion in the derivation is a couple  $(i, j)$  with  $i < j$  such that the rule applied at step  $i$  is in  $R_2$  whereas the rule applied at step  $j$  is in  $R_1$ . We will prove that the derivation can be “sorted” and that  $u \xrightarrow{*}_{R_1} \circ \xrightarrow{*}_{R_2} v$  by induction of the number of inversions in the derivation. If the number of inversions is 0, by definition,  $u \xrightarrow{*}_{R_1} \circ \xrightarrow{*}_{R_2} v$ . Otherwise, there is an inversion: this implies that there exists a step  $i$  such that the rule applied at step  $i$  is in  $R_2$  whereas the rule applied at step  $i + 1$  is in  $R_1$ . By what precedes, we can permute these two steps and we get a derivation  $u \xrightarrow{*}_R v$  whose number of inversions is strictly smaller: so, by induction,  $u \xrightarrow{*}_{R_1} \circ \xrightarrow{*}_{R_2} v$ . ◀

## B

 Proof of Lemmas 16, 17, 18

**Proof of Lemma 16.** By Theorem 4 we just need to prove that if  $p \sqsubseteq_R^f Q$  then  $p \sqsubseteq_R^\omega Q$ .  $D_R(p)$  is a regular language closed by  $R$  and contains  $p$ . Then, by Theorem 5, if  $p \sqsubseteq_R^f Q$ ,  $D_R(p)$  intersects with  $Q$  and so, there exists  $q$  in  $Q$  such that  $p \xrightarrow{*}_R q$ ; then, by Theorem 2,  $p \sqsubseteq_R^\omega q$  and so,  $p \sqsubseteq_R^\omega Q$ . ◀

**Proof of Lemma 17.** We take the rewrite system  $R$  of Section 4.2. We let  $P = \Sigma^+$  and  $Q = a^+ + b^+$ . We define  $B$  as the set  $\{u \mid u \neq \varepsilon \wedge |u|_a = |u|_b\}$ . For every  $u \notin B$ , we either have that  $|u|_a > |u|_b$  or  $|u|_a < |u|_b$ . In the first case,  $u \xrightarrow{*}_R a^k$  for  $k = |u|_a - |u|_b$  and, in the second case,  $u \xrightarrow{*}_R b^k$  for  $k = |u|_b - |u|_a$ . So for every  $u \notin B$ , we have that  $u \sqsubseteq_R^\omega a^+ + b^+$  and thus  $u \sqsubseteq_R^f a^+ + b^+$ . In contrast, we have that  $D_R(B) = B$  and  $B \cap a^+ + b^+ = \emptyset$ . Therefore for every  $u \in B$ , we do not have  $u \sqsubseteq_R^\omega a^+ + b^+$ . Thus, we do not have that  $\Sigma^+ \sqsubseteq_R^\omega b^+ + a^+$ .

However, for every  $u \in B$ , let  $n = |u|_a$ , as  $u \xrightarrow{*}_R a^n b^n$  we have that  $u \sqsubseteq_R^f a^n b^n$ . We have seen Section 4.2 that  $a^n b^n \sqsubseteq_R^f b^m$  for some  $m > 0$ . This shows that  $\Sigma^+ \sqsubseteq_R^f a^+ + b^+$ . ◀

**Proof of Lemma 18.** By Theorem 4 we just need to prove that if  $Q_1 \sqsubseteq_R^f Q_2$  then  $Q_1 \sqsubseteq_R^\omega Q_2$ .  $K = \Sigma^*/A_R(Q_2)$  is a regular language closed under  $R$  that does not intersect with  $Q_2$ . From Theorem 5, if  $Q_1 \sqsubseteq_R^f Q_2$ ,  $K$  does not intersect with  $Q_1$ ; then  $Q_1 \subseteq A_R(Q_2)$  and by Theorem 4  $Q_1 \sqsubseteq_R^\omega Q_2$ . ◀



### C Proof of Proposition 22

**Proof.** We only need to prove that  $q \sqsubseteq_R^f Q$  implies  $q \sqsubseteq_R^\omega Q$ . Take a path  $q$  and a regular set of paths  $Q$  such that there is no  $p$  in  $Q$  with  $q \xrightarrow{*}_R p$ . We will prove the existence of a regular language  $K$  closed under  $R$  containing  $q$  and not intersecting with  $Q$ .

Let  $q_1 p_1 \dots q_n p_n$  be the unique decomposition of  $q$  such that  $q_1 \in \Sigma_1^*$ ,  $p_n \in \Sigma_2^*$ ,  $q_2, \dots, q_n \in \Sigma_1^+$  and  $p_1, \dots, p_{n-1} \in \Sigma_2^+$ .

Let  $P = \Sigma_1^*(\Sigma_2^+ \Sigma_1^+)^{n-1} \Sigma_2^*$ . If  $Q \cap P$  is empty, we can choose  $P$  for  $K$ : it is easy to check that it satisfies all the conditions. Otherwise,  $Q \cap P$  is a non empty regular language included in  $P$ .

We use the following property: given three regular sets  $N, A, B$ , if  $N \subseteq AB$ , then  $N = \bigcup_{i \in I} A_i B_i$  where the  $A_i$  (resp.  $B_i$ ) are all regular, all included in  $A$  (resp.  $B$ ) and  $I$  is finite. Indeed it is easy to check that  $N = \bigcup_{u \in A} ([u] \cap A) \cdot (u^{-1} N \cap B)$ , where  $[u] = \{v \mid u^{-1} N = v^{-1} N\}$ . The number of distincts  $[u]$  and  $u^{-1} N$  is finite as  $N$  is regular, so we get the required finite decomposition.

Iterating the aforementioned property, we get that  $Q \cap P$  can be decomposed in a finite union of products of regular languages  $\bigcup_{i \in I} A_{1,i} B_{1,i} \dots A_{n,i} B_{n,i}$  for some finite  $I$ , where  $A_{1,i} \subseteq \Sigma_1^*$ ,  $B_{n,i} \subseteq \Sigma_2^*$ ,  $A_{j,i} \subseteq \Sigma_1^+$  for  $j \neq 1$ ,  $B_{j,i} \subseteq \Sigma_2^+$ , for  $j \neq n$ .

As, by hypothesis, there is no  $p$  in  $Q$  so that  $q \xrightarrow{*}_R p$ ,  $Q \cap P \cap D_R(q) = \emptyset$ . Thus, for every  $i$  in  $I$ ,  $A_{1,i} B_{1,i} \dots A_{n,i} B_{n,i} \cap D_R(q)$  is empty and so there exists  $j$  such that  $A_{j,i} \cap D_{R_1}(q_j) = \emptyset$  or  $B_{j,i} \cap D_{R_2}(p_j) = \emptyset$ . Since  $R_1$  (resp.  $R_2$ ) is finitely controllable, there is a regular language  $K_1$  (resp.  $K_2$ ) closed under  $R_1$  (resp.  $R_2$ ) containing  $D_{R_1}(q_j)$  (resp.  $D_{R_2}(p_j)$ ) and not intersecting with  $A_{j,i}$  (resp.  $B_{j,i}$ ).

Define  $L_i$  as:

- $L_i = \Sigma_1^* (\Sigma_2^+ \Sigma_1^+)^{j-2} \Sigma_2^+ (K_1 \cap \Sigma_1^+) (\Sigma_2^+ \Sigma_1^+)^{n-j} \Sigma_2^*$   
in case  $A_{j,i} \cap D_{R_1}(q_j) = \emptyset$ ,
- $L_i = \Sigma_1^* (\Sigma_2^+ \Sigma_1^+)^{j-1} (K_2 \cap \Sigma_2^+) (\Sigma_1^+ \Sigma_2^+)^{n-j-1} \Sigma_1^+ \Sigma_2^*$   
otherwise (in this case  $B_{j,i} \cap D_{R_2}(p_j) = \emptyset$ ).

It is easy to check that in both cases  $L_i$  is regular, closed under  $R$ , contains  $q$ , does not intersect with  $A_{1,i} B_{1,i} \dots A_{n,i} B_{n,i}$ .

Define  $K$  as  $\bigcap_{i \in I} L_i$ :  $K$  is regular, closed under  $R$ , contains  $q$ , does not intersect with  $Q$ . By Theorem 5, we obtain that it is not the case that  $q \sqsubseteq_R^f Q$ . ◀

### D Proof of Lemma 23

**Proof.** Our proof relies on undecidability of emptiness of the intersection of recursive sets of numbers. We represent sets of numbers as one letter languages. For  $k \neq 0$ , we let  $\mathbf{p}_k$  to be the  $k^{\text{th}}$  prime number and, given a letter  $a$  and a set of numbers  $\mathcal{N}$ , we write  $\mathcal{P}_a(\mathcal{N})$  for the language  $\{a^{\mathbf{p}_n} \mid n \in \mathcal{N}\}$ . We write  $\mathbf{P}_a$  for the language  $\{a^p \mid p \text{ prime number}\}$  and  $\mathbf{P}_a^c$  for  $a^* - \mathbf{P}_a$ .

First, let us notice that if  $L$  is regular and  $L$  is included in  $\mathbf{P}_a$ , then  $L$  is finite. Indeed, by a pumping argument, we get that if  $L$  is an infinite regular language included in  $\Sigma^*$ , there exists  $n \geq 0, m > 0$  such that  $a^{n+pm}$  belongs to  $L$  for any integer  $p$ . But, then  $a^{n+(n+2m+2)m}$  belongs to  $L$ . As  $n + (n + 2m + 2) * m = (n + 2m) * (m + 1)$  is not prime,  $L$  is not included in  $\mathbf{P}_a$ .

Let  $\mathcal{N}_1$  and  $\mathcal{N}_2$  two infinite arbitrary recursive sets of numbers. We let  $\mathcal{L}_1 = \mathbf{P}_a^c \cup \mathcal{P}_a(\mathcal{N}_1)$ ,  $\mathcal{L}_2 = \mathcal{P}_a(\mathcal{N}_2)$ . We have that  $\mathcal{N}_1 \cap \mathcal{N}_2 = \emptyset$  iff  $\mathcal{L}_1 \cap \mathcal{L}_2 = \emptyset$ . If  $R$  is a regular language containing  $\mathcal{L}_1$ ,  $a^*/R$  is included in  $\mathbf{P}_a$ , and then, from what precedes, is finite; so,  $R$  contains

all, but finitely, many elements of  $a^*$  and as  $\mathcal{L}_2$  is infinite, we must have  $\mathcal{L}_2 \cap R \neq \emptyset$ . In other words,  $\mathcal{L}_1$  and  $\mathcal{L}_2$  cannot be separated by a regular set. However deciding  $\mathcal{L}_1 \cap \mathcal{L}_2 = \emptyset$  is equivalent to deciding  $\mathcal{N}_1 \cap \mathcal{N}_2 = \emptyset$ .  $\blacktriangleleft$

## E Proof of Lemma 24

**Proof.** We prove below the different items, the second one being the most technical.


1. If  $L_1 \cap L_2$  is non empty, there exists  $w$  such that  $s_1 \xrightarrow{*}_{R_1} w \xrightarrow{*}_B \bar{w} \xrightarrow{*}_{R_2} s_2$ , so  $s_1 \xrightarrow{*}_R s_2$ . Let  $(u, v)$  in  $\Delta^+ / \{\#_l, \#_r\}^* \times \Delta^+$ :  $u$  can be decomposed in  $u_1 x u_2$  with  $x \notin \{\#_l, \#_r\}$ . Then,  $u \rightarrow_{R_\#} u_1 \#_l s_1 \#_r u_2 \xrightarrow{*}_R u_1 \#_l s_2 \#_r u_2 \rightarrow_{R_g} u_1 \#_l g \#_r u_2 \xrightarrow{*}_{R_g} g$ . As  $g$  can generate any word of  $\Delta^+$ , we have  $u \xrightarrow{*}_R v$ .
2. Suppose  $\#_l s_1 \#_r \xrightarrow{*}_R \#_l s_2 \#_r$ . We first prove that  $\#_l s_1 \#_r \xrightarrow{*}_{R_1 \cup B \cup R_2 \cup R_\#} m_1 \#_l s_2 \#_r m_2$ . If no  $g$  occurs in the derivation, the derivation is in  $R_1 \cup B \cup R_2 \cup R_\#$  and satisfies the requirements. Otherwise, let us consider  $i_g$ , the first step of the derivation where  $g$  occurs. As  $g$  is only produced by  $\#_l s_2 \#_r \rightarrow g$  or by a rule containing  $g$  in its l.h.s., the derivation truncated at its  $i_g - 1$  first steps is of the form  $\#_l s_1 \#_r \xrightarrow{*}_{R_1 \cup B \cup R_2 \cup R_\#} m_1 \#_l s_2 \#_r m_2$  for some  $m_1, m_2$ . Now, let us prove that by induction of the length of the derivation the following property: Let  $u$  in  $(\Sigma_1 \cup \Sigma_2)^*$ ; If  $\#_l s_1 \#_r \xrightarrow{*}_{R_1 \cup B \cup R_2 \cup R_\#} m_1 \#_l u \#_r m_2$  for some  $m_1, m_2$ , there exists a derivation  $\#_l s_1 \#_r \xrightarrow{*}_{R_1 \cup B \cup R_2} \#_l u \#_r$ . If the derivation is of length 1, either  $\#_l s_1 \#_r \rightarrow_{R_1} \#_l u \#_r$  or  $\#_l s_1 \#_r \rightarrow_{R_\#} \#_l \#_l s_1 \#_r \#_r$  and  $u = s_1$ . In both cases,  $\#_l s_1 \#_r \xrightarrow{\leq 1}_{R_1} \#_l u \#_r$ . Let us now suppose that the property is true for derivations of length  $n$  and let a derivation of length  $n + 1$ : If  $\#_l u \#_r$  is not concerned by the last rewriting step, we have  $\#_l s_1 \#_r \xrightarrow{n}_{R_1 \cup B \cup R_2 \cup R_\#} m'_1 \#_l u \#_r m'_2$  for some  $m'_1, m'_2$  and we have the property by induction. If  $\#_l u \#_r$  is concerned by the last step, as  $u$  is in  $(\Sigma_1 \cup \Sigma_2)^*$ , either the last step uses a rule  $x \rightarrow_{R_\#} \#_l s_1 \#_r$  and  $u = s_1$ , so the property is trivial or the derivation is of the form  $\#_l s_1 \#_r \xrightarrow{n}_{R_1 \cup B \cup R_2 \cup R_\#} m_1 \#_l u' \#_l m_2 \rightarrow_{R_1 \cup B \cup R_2} m_1 \#_l u \#_r m_2$ . Then by induction,  $\#_l s_1 \#_r \xrightarrow{*}_{R_1 \cup B \cup R_2} \#_l u' \#_r \rightarrow_{R_1 \cup B \cup R_2} \#_l u \#_r$ . Applying the property to  $\#_l s_1 \#_r \xrightarrow{*}_{R_1 \cup B \cup R_2 \cup R_\#} m_1 \#_l s_2 \#_r m_2$ , we get that there is a derivation  $\#_l s_1 \#_r \xrightarrow{*}_{R_1 \cup B \cup R_2} \#_l s_2 \#_r$ . By adapting the reasonment already used in Lemma 12, there exists a derivation  $\#_l s_1 \#_r \xrightarrow{*}_{R_1} \#_l m \#_r \xrightarrow{*}_B \#_l \bar{m} \#_r \xrightarrow{*}_{R_2} \#_l s_2 \#_r$ . But then  $m$  belongs to  $L_1 \cap L_2$  that would not be empty.
3. Every regular language closed under  $R$  containing  $\#_g s_1 \#_d$  contains  $\#_g L_1 \#_d$  and then intersects with  $\#_g L_2 \#_d$  as  $L_1$  and  $L_2$  are not regularly separable. So, by Corollary 10,  $\#_g s_1 \#_d \sqsubseteq_R^f \#_g s_2 \#_d$ .
4. If  $u \in \{\#_g, \#_d\}^*$ , then  $D_R(u) = \{u\}$  is a regular language closed under  $R$ . By Corollary 10,  $u \sqsubseteq_R^f v$  iff  $u = v$  and, therefore, iff  $u \xrightarrow{*}_R v$ .
5. Let  $u \in \Delta^+ / \{\#_l, \#_r\}^*$ : we have  $u = u_1 x u_2$  with  $x \in \Delta$ . Therefore,  $u \rightarrow_{R_\#} u_1 \#_l s_1 \#_r u_2$ , so  $u \sqsubseteq_R^f u_1 \#_l s_1 \#_r u_2$ . As  $\#_l s_1 \#_r \sqsubseteq_R^f \#_l s_2 \#_r$ ,  $u \sqsubseteq_R^f u_1 \#_l s_2 \#_r u_2 \sqsubseteq_R^f u_1 g u_2 \sqsubseteq_R^f v$  for any  $v$ . In case  $u \in \{\#_l, \#_r\}^*$ , we have already seen that that  $u \sqsubseteq_R^f v$  iff  $u = v$ . In a nutshell,  $u \sqsubseteq_R^f v$  iff  $u \in \Delta^+ / \{\#_l, \#_r\}^*$  or  $u = v$ .  $\blacktriangleleft$





# Computing Data Distribution from Query Selectivities

Pankaj K. Agarwal  


Department of Computer Science, Duke University, Durham, NC, USA

Rahul Raychaudhury 

Department of Computer Science, Duke University, Durham, NC, USA

Stavros Sintos  

Department of Computer Science, University of Illinois at Chicago, IL, USA

Jun Yang  

Department of Computer Science, Duke University, Durham, NC, USA

---

## Abstract

---

We are given a set  $\mathcal{Z} = \{(R_1, s_1), \dots, (R_n, s_n)\}$ , where each  $R_i$  is a *range* in  $\mathbb{R}^d$ , such as rectangle or ball, and  $s_i \in [0, 1]$  denotes its *selectivity*. The goal is to compute a small-size *discrete data distribution*  $\mathcal{D} = \{(q_1, w_1), \dots, (q_m, w_m)\}$ , where  $q_j \in \mathbb{R}^d$  and  $w_j \in [0, 1]$  for each  $1 \leq j \leq m$ , and  $\sum_{1 \leq j \leq m} w_j = 1$ , such that  $\mathcal{D}$  is the most *consistent* with  $\mathcal{Z}$ , i.e.,  $\text{err}_p(\mathcal{D}, \mathcal{Z}) = \frac{1}{n} \sum_{i=1}^n |s_i - \sum_{j=1}^m w_j \cdot \mathbb{1}(q_j \in R_i)|^p$  is minimized. In a database setting,  $\mathcal{Z}$  corresponds to a workload of range queries over some table, together with their observed selectivities (i.e., fraction of tuples returned), and  $\mathcal{D}$  can be used as compact model for approximating the data distribution within the table without accessing the underlying contents.

In this paper, we obtain both upper and lower bounds for this problem. In particular, we show that the problem of finding the best data distribution from selectivity queries is NP-complete. On the positive side, we describe a Monte Carlo algorithm that constructs, in time  $O((n + \delta^{-d})\delta^{-2} \text{polylog } n)$ , a discrete distribution  $\hat{\mathcal{D}}$  of size  $O(\delta^{-2})$ , such that  $\text{err}_p(\hat{\mathcal{D}}, \mathcal{Z}) \leq \min_{\mathcal{D}} \text{err}_p(\mathcal{D}, \mathcal{Z}) + \delta$  (for  $p = 1, 2, \infty$ ) where the minimum is taken over all discrete distributions. We also establish conditional lower bounds, which strongly indicate the infeasibility of relative approximations as well as removal of the exponential dependency on the dimension for additive approximations. This suggests that significant improvements to our algorithm are unlikely.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Computational geometry

**Keywords and phrases** selectivity queries, discrete distributions, Multiplicative Weights Update, eps-approximation, learnable functions, depth problem, arrangement

**Digital Object Identifier** 10.4230/LIPIcs.ICDT.2024.18

**Related Version** *Full Version*: <https://arxiv.org/abs/2401.06047> [1]

## 1 Introduction

The *selectivity* of a selection query on a set of objects in a database is the probability of a random object in the database satisfying the query predicate. A key step in query optimization, selectivity estimation is used by databases for estimating costs of alternative query processing plans and picking the best one. Consequently, selectivity estimation has been studied extensively in the last few decades [31, 36, 43, 44, 46].

Historically, selectivity estimation has been data-driven. These approaches construct, or dynamically maintain, a small-size synopsis of the data distribution using histograms or random samples that minimize estimation error. While these methods work well in low dimensions, they suffer from the curse of dimensionality. As a result, interest in learning-based methods for selectivity estimation has been growing over the years [24, 32, 33, 34, 38, 40]. Many different methods have been proposed that work with the data distribution, observed



© Pankaj K. Agarwal, Rahul Raychaudhury, Stavros Sintos, and Jun Yang;  
licensed under Creative Commons License CC-BY 4.0

27th International Conference on Database Theory (ICDT 2024).

Editors: Graham Cormode and Michael Shekelyan; Article No. 18; pp. 18:1–18:20

Leibniz International Proceedings in Informatics



Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

selectivities from query workloads, or a combination of both. At a high-level, many of these techniques build a model of the underlying data distribution and use it to answer queries. While they work very well in practice, often outperforming their traditional counterparts, a theoretical understanding of this line of work is missing. This leads to the natural question, whether selectivity can be learned efficiently from a small sample of query selectivities alone, without access to the data distribution. Hu et al. [27] formalize the learnability of the selectivity estimation problem in this setting. They use the agnostic-learning framework [25], an extension of the classical PAC learning framework for real-valued functions, where one is given a set of sample queries from a fixed query distribution and their respective selectivities (the *training set*), and the goal is to efficiently construct a data distribution so that the selectivity of a new query from the same query distribution can be answered with high accuracy. They show that for a wide class of range queries, the selectivity query can be learned within error  $\varepsilon \in (0, 1)$  with probability at least  $1 - \delta$  using a training set of size  $\varepsilon^{-O(1)} \log \delta^{-1}$ , where the exponent of  $\varepsilon$  depends on the query type; see [27] for a precise statement of their results. Informally, learnability implies that performance of a model on the training set generalizes to unseen queries from the same distribution. This reduces the task of learning to finding a (model of the) data distribution that best fits the training data i.e. *Empirical Risk Minimization (ERM)*. Although Hu et al. [27] prove a sharp bound on the sample complexity, their algorithm for ERM takes prohibitively long and produces a data distribution of large size. They also present fast heuristics to construct small-size data distributions, but they do not provide any guarantee on the performance with respect to the best data distribution fitting the training set. This raises the question of how to develop a provably efficient and effective algorithm for constructing the best data distribution (in a given family) from a training set.

Note that the size of the distribution computed by [27], can further be reduced to  $O(\varepsilon^{-2})$  by choosing an  $\varepsilon$ -approximation, with an increase of  $\varepsilon$  in the error; see [23] and Section 3 below. However, the paper aims at computing a small-size distribution (whose performance is comparable to the best data distribution) directly and efficiently, without constructing a large-size distribution first. For this problem, we obtain hardness results as well as efficient algorithms.

**Problem Statement.** A *range space*  $\Sigma = (\mathcal{X}, \mathcal{R})$  comprises a set of *objects*  $\mathcal{X}$  and a collection of subsets  $\mathcal{R} \subseteq 2^{\mathcal{X}}$  called *ranges*. In this paper,  $\mathcal{X}$  is a finite set of points, and each range  $R \in \mathcal{R}$  corresponds to a query that returns the subset of objects that fall within a simple geometric region such as a rectangle (corresponding to an orthogonal range query), a ball (neighborhood range query), or a half-space (query with a linear constraint). We will not distinguish between a region  $R$  and  $R \cap \mathcal{X}$ , so with a slight abuse of notation, we will use  $\mathcal{R}$  to denote a set of geometric regions such as a set of rectangles or balls.

A *discrete distribution*  $\mathcal{D} = \{(p_1, w_1), \dots, (p_m, w_m)\}$  is defined by a finite set of points and their associated probabilities, where each  $p_i$  is a point in  $\mathbb{R}^d$  for some constant  $d \geq 1$ , each  $w_i > 0$ , and  $\sum_{i=1}^m w_i = 1$ . We refer to the point set  $\{p_1, \dots, p_m\}$  as the *support* of  $\mathcal{D}$  and denote it by  $\text{supp}(\mathcal{D})$ . The *size* of a discrete distribution  $\mathcal{D}$  is defined as the size of its support and is denoted as  $|\mathcal{D}|$ . Let  $\mathbb{D}$  denote the family of all discrete distributions, and let  $\mathbb{D}_k$  be the family of discrete distributions of size at most  $k$ . Given a range  $R \subseteq \mathcal{R}$ , let  $s_{\mathcal{D}}(R) = \sum_{i=1}^m \mathbb{1}(p_i \in R)w_i$  denote the *selectivity* of  $R$  over  $\mathcal{D}$ , which is the probability for a random point drawn from  $\mathcal{D}$  to lie in  $R$  (or the total measure of  $\mathcal{D}$  inside  $R$ ).

In this paper, our goal is to learn a (discrete) distribution from the selectivities of range queries. For ease of exposition, we will describe our results for rectangles (orthogonal ranges), although our techniques extend to other natural range spaces.

Let  $\mathcal{Z} = \{z_1, \dots, z_n\}$  be a set of *training samples*, where  $z_i = (R_i, s_i)$ ,  $R_i$  is an axis-aligned rectangle (orthogonal range) in  $\mathbb{R}^d$ , and  $s_i \in [0, 1]$  is its observed selectivity.<sup>1</sup> Let  $\mathcal{R} = \{R_1, \dots, R_n\}$  denote the set of ranges in  $\mathcal{Z}$ . For a discrete distribution  $\mathcal{D} \in \mathbb{D}$  and for  $p \geq 1$  we define the ( $\ell_p$ ) empirical error to be

$$\text{err}_p(\mathcal{D}, \mathcal{Z}) = \frac{1}{n} \sum_{i=1}^n |s_{\mathcal{D}}(R_i) - s_i|^p, \text{ and for } p = \infty, \text{err}_{\infty}(\mathcal{D}, \mathcal{Z}) = \max_{1 \leq i \leq n} |s_{\mathcal{D}}(R_i) - s_i|. \quad (1)$$

We focus on  $p = 1, 2$ , and  $\infty$ .

Let  $\alpha_p^*(\mathcal{Z}) = \min_{\mathcal{D} \in \mathbb{D}} \text{err}_p(\mathcal{D}, \mathcal{Z})$  denote the minimum error achievable for all distributions in the family. Given  $\mathcal{Z}$ , our goal is to compute a discrete distribution  $\hat{\mathcal{D}}_p$  with  $\text{err}_p(\hat{\mathcal{D}}_p, \mathcal{Z}) = \alpha_p^*$ . As argued in [27], such a discrete distribution of size  $O(n^d)$  can be computed in  $n^{O(d)}$  time. However, this distribution is too large even for moderate values of  $n$  and  $d$ , so we are interested in computing a smaller size distribution at the cost of a slight increase in the empirical error. Hence, our problem becomes the following: given  $\mathcal{Z}$  and some  $\delta \in [0, 1]$ , compute a small-size distribution  $\mathcal{D}$ , ideally of size that depends on  $\delta$  and independent of  $n$ , such that  $\text{err}_p(\mathcal{D}, \mathcal{Z}) \leq \alpha_p^* + \delta$ . Alternatively, given a size budget  $k$ , compute a distribution  $\hat{\mathcal{D}} \in \mathbb{D}_k$  such that  $\text{err}_p(\hat{\mathcal{D}}, \mathcal{Z}) = \min_{\mathcal{D} \in \mathbb{D}_k} \text{err}_p(\mathcal{D}, \mathcal{Z})$ .

**Our results.** We present both negative and positive results for the data-distribution<sup>2</sup> learning problem. On the negative side in Section 5, we prove that the problem is NP-complete even for rectangles in  $\mathbb{R}^2$ . Namely, given  $\mathcal{Z}$  where  $\mathcal{R}$  is a set of rectangles in  $\mathbb{R}^2$ ,  $p \in \{1, 2, \infty\}$ , and two parameters  $\delta \in [0, 1]$  and  $k \geq 1$ , the problem of determining whether there is a data distribution  $\mathcal{D}$  of size  $k$  such that  $\text{err}_p(\mathcal{D}, \mathcal{Z}) \leq \delta$  is NP-hard.

On the positive side, we focus on designing an efficient algorithm for constructing a small-size distribution with additive-approximation error. We present a Monte Carlo algorithm that computes, with high probability, a discrete distribution  $\tilde{\mathcal{D}}$  of size  $O(\delta^{-2})$  with  $\text{err}_p(\tilde{\mathcal{D}}, \mathcal{Z}) \leq \alpha_p^* + \delta$ , for  $p \in \{1, 2, \infty\}$ , in  $O(n\delta^{-2} \log n + \delta^{-d-2} \log^3 n)$  time (exact time complexity is given in Lemma 8). Starting with  $p = 1$  ( $\ell_1$  empirical error), we first present in Section 2 a basic algorithm that maps our problem to a linear program (LP) with  $O(n)$  constraints and  $O(n^d)$  variables. We show how the Multiplicative-Weight-Update (MWU) method [5] can be used to solve this LP. A naive implementation of the MWU of this algorithm takes  $\Omega(n^d)$  time. Next, in Section 3, we exploit underlying geometry in two ways to solve this LP involving exponential number of variables efficiently, by representing it implicitly. First, we give a geometric interpretation to the main step of the MWU method: we map a maximization problem with  $O(n^d)$  variables to the problem of finding the weighted deepest point in an arrangement of  $n$  rectangles in  $\mathbb{R}^d$ . The best algorithm to solve this geometric problem takes  $O(n^{d/2})$  time [9], which would allow one to improve the running time to  $O(\delta^{-2} n^{d/2} \log n)$ . But this is also expensive. Second, we use the notion of  $\varepsilon$ -approximation to quickly compute an approximately deepest point in a weighted set of rectangles efficiently, in time  $O(\delta^{-d} \log n)$ .

<sup>1</sup> Note that we do not assume the training set  $\mathcal{Z}$  to be consistent with any distribution in  $\mathbb{D}$ , or any distribution in general; i.e., there might not exist any distribution  $\mathcal{D}$  such that  $s_i$  reflects the selectivity of  $R_i$  for every  $1 \leq i \leq n$ . This flexibility allows us to model settings where the query workload was executed on an evolving database instance, and the observed selectivities on different concrete instances may not be consistent with each other.

<sup>2</sup> In this paper we focus on discrete distributions. For simplicity, sometimes we use the term data distribution instead of discrete distribution.

In Section 4 we extend our algorithm to more general settings. We show that our near-linear time algorithm works for the  $\ell_\infty$  and the  $\ell_2$  empirical error. Furthermore, we show that our algorithm can be extended to other ranges such as balls and halfspaces in  $\mathbb{R}^d$ .

In Section 5, we also give conditional lower bounds that indicate that avoiding the exponential dependency on  $d$  is not possible even for additive approximations. This makes meaningful improvements to our algorithm unlikely. We also give conditional lower bounds for a variant of our problem, allowing an arbitrary relative approximation factor on the size of the distribution or an arbitrary relative approximation factor on the error of the returned distribution. Our conditional hardness results are based on the  $\text{FPT} \neq W[1]$  conjecture; see [14] for the definition. Finally, we also show that when the distribution size is fixed, any relative approximation is NP-hard.

## 2 Basic Algorithm

In this section we present our basic algorithm for computing a small-size discrete distribution that (approximately) minimizes the  $\ell_1$  empirical error. For simplicity, let  $\text{err}(\cdot, \cdot)$  and  $\alpha^*$  denote  $\text{err}_1(\cdot, \cdot)$  and  $\alpha_1^*$ , respectively. Given a training set  $\mathcal{Z}$  and a parameter  $\delta \in (0, 1)$ , it computes a discrete distribution  $\mathcal{D}^*$  of size  $O(\delta^{-2} \log n)$  such that  $\text{err}(\mathcal{D}^*, \mathcal{Z}) = \alpha^* + \delta$ . At the heart, there is a decision procedure `ISFEASIBLE` that, given  $\mathcal{Z}$  and parameters  $\alpha, \delta \in (0, 1)$ , returns a distribution  $\mathcal{D}^*$  with  $\text{err}(\mathcal{D}^*, \mathcal{Z}) \leq \alpha + \delta/2$  if  $\alpha \geq \alpha^*$  and returns `NO` if  $\alpha < \alpha^*$ .

We do not know the value of  $\alpha^*$ , so we perform a binary search on the value of  $\alpha$ . In particular, let  $E = \{\frac{\delta}{2}, (1 + \frac{\delta}{2})\frac{\delta}{2}, (1 + \frac{\delta}{2})^2\frac{\delta}{2}, \dots, 1\}$  be a discretization of the range  $[0, 1]$ , and let  $\alpha_j$  be the  $j$ -th value of  $E$ . Suppose the binary search currently guesses  $\alpha_i$  to be the current guess of  $\alpha^*$ . If `ISFEASIBLE`( $\mathcal{Z}, \delta, \alpha_i$ ) returns a distribution  $\mathcal{D}$ , we continue the binary search for values less than  $\alpha_i$  in  $E$ . Otherwise, we continue the binary search for values greater than  $\alpha_i$  in  $E$ . At the end of the binary search, we return the last distribution  $\mathcal{D}^*$  that `ISFEASIBLE` found. Assuming the correctness of the decision procedure, the algorithm returns the desired distribution in  $\log \frac{1}{\delta}$  iterations. If  $\alpha^* \leq \delta/2$ , then `ISFEASIBLE`( $\mathcal{Z}, \delta, \alpha_1$ ) returns a distribution  $\mathcal{D}^*$  such that  $\text{err}(\mathcal{D}^*, \mathcal{Z}) \leq \alpha_1 + \delta/2 = \delta \leq \alpha^* + \delta$ . In any other case, without loss of generality assuming that  $\alpha_{i-1} < \alpha^* \leq \alpha_i$ , we have  $\alpha_i \leq (1 + \delta/2)\alpha^*$ . By definition, `ISFEASIBLE`( $\mathcal{Z}, \delta, \alpha_i$ ) returns a distribution  $\mathcal{D}_i$  such that  $\text{err}(\mathcal{D}_i, \mathcal{Z}) \leq \alpha_i + \delta/2$ . Hence,

$$\text{err}(\mathcal{D}^*, \mathcal{Z}) \leq \text{err}(\mathcal{D}_i, \mathcal{Z}) \leq \alpha_i + \delta/2 \leq (1 + \delta/2)\alpha^* + \delta/2 \leq \alpha^* + \delta.$$

We now describe the decision procedure `ISFEASIBLE`.

### 2.1 Decision procedure

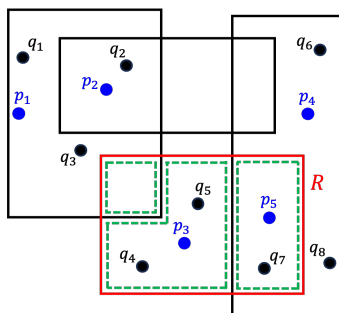
Let  $\mathcal{Z}, \delta$ , and  $\alpha$  be as defined above. The decision problem can be formulated as follows:

(FP1)  $\exists? \mathcal{D} \in \mathbb{D}$  s.t.

$$\begin{aligned} \frac{1}{n} \sum_{i=1}^n u_i &\leq \alpha \\ |s_{\mathcal{D}}(R_i) - s_i| &\leq u_i \quad \text{for } i = 1 \dots n \\ u_i &\in [0, 1] \quad \text{for } i = 1 \dots n. \end{aligned}$$

Here  $u_i$  models the error in the selectivity of  $R_i$ . A challenge in solving the above decision problem is determining the candidate set of points in  $\text{supp}(\mathcal{D})$ . The problem as stated is infinite-dimensional. Our next lemma suggests how to reduce it to a finite-dimensional problem by constructing a finite set of candidate points for  $\text{supp}(\mathcal{D})$ .





■ **Figure 1** The (black) points  $q_i$  represent points from the underlying data distribution  $\mathcal{D}$  while (blue) points  $p_\tau \in \mathcal{P}$ . The (green) dashed segments show three cells of the arrangement in rectangle  $R$ . The weights of points in  $\mathcal{P}$  are:  $w(p_1) = w_1 + w_3$ ,  $w(p_2) = w_2$ ,  $w(p_3) = w_4 + w_5$ ,  $w(p_4) = w_6 + w_8$ ,  $w(p_5) = w_7$ .

The *arrangement* of  $\mathcal{R}$ , denoted  $\mathcal{A}(\mathcal{R})$ , is a partitioning of  $\mathbb{R}^d$  into contiguous regions called *cells* such that for every ( $d$ -dimensional) cell  $\tau$  in the arrangement,  $\tau$  lies in the same subset of  $\mathcal{R}$ . For each  $d$ -dimensional cell  $\tau \in \mathcal{A}(\mathcal{R})$ , we choose an arbitrary point  $p_\tau$  in the interior of  $\tau$ . Let  $\mathcal{P} = \{p_\tau \mid \tau \in \mathcal{A}(\mathcal{R})\}$  be the set of candidate points. Note that  $\mathcal{A}(\mathcal{R})$  has  $O(n^d)$  cells, and it can be computed in  $O(n^d \log n)$  time [3]. Therefore,  $|\mathcal{P}| = O(n^d)$  and  $\mathcal{P}$  can be computed in  $O(n^d \log n)$  time.

► **Lemma 1.** *For any discrete distribution  $\mathcal{D} \in \mathbb{D}$ , there is another discrete distribution  $\mathcal{D}'$  such that  $\text{supp}(\mathcal{D}') \subseteq \mathcal{P}$  and  $\text{err}(\mathcal{D}, \mathcal{Z}) = \text{err}(\mathcal{D}', \mathcal{Z})$ .*

**Proof.** For each  $d$ -dimensional cell  $\tau \in \mathcal{A}(\mathcal{R})$ , let

$$w_\tau = \sum_{p_i \in \text{supp}(\mathcal{D})} \mathbb{1}(p_i \in \tau) w_i$$

be the total weight of points of  $\text{supp}(\mathcal{D})$  that lie in  $\tau$ . We define

$$\mathcal{D}' = \{(p_\tau, w_\tau) \mid \tau \in \mathcal{A}(\mathcal{R}), w_\tau > 0\}.$$

See also Figure 1. By definition, for any rectangle  $R \in \mathcal{R}$ , the cells of  $\mathcal{A}(\mathcal{R})$  lying inside  $R$  induce a partitioning of  $R$ . Let  $\mathcal{A}(\mathcal{R} \mid R)$  denote this partitioning of  $R$  into cells. Then

$$\begin{aligned} s_{\mathcal{D}'}(R) &= \sum_{p_\tau \in \mathcal{P}} \mathbb{1}(p_\tau \in R) w_\tau = \sum_{\tau \in \mathcal{A}(\mathcal{R} \mid R)} w_\tau = \sum_{\tau \in \mathcal{A}(\mathcal{R} \mid R)} \sum_{p_i \in \tau \cap \text{supp}(\mathcal{D})} w_i \\ &= \sum_{p_i \in \text{supp}(\mathcal{D})} \mathbb{1}(p_i \in R) w_i = s_{\mathcal{D}}(R). \end{aligned}$$

Hence,  $\text{err}(\mathcal{D}, \mathcal{Z}) = \text{err}(\mathcal{D}', \mathcal{Z})$ . ◀

► **Remark.** We note that the choice of point  $p_\tau$  in each cell  $\tau \in \mathcal{A}(\mathcal{R})$  is arbitrary; one can use any point in  $\tau$  as its representative point.

In view of Lemma 1, it suffices to restrict  $\text{supp}(\mathcal{D})$  to be a subset of  $\mathcal{P}$ . We order the cells of  $\mathcal{A}(\mathcal{R})$  arbitrarily and let  $p_j$  denote the point chosen from the  $j$ -th cell, so  $\mathcal{P} = \{p_1, \dots, p_m\}$  where  $m = |\mathcal{P}|$ . We introduce a real variable  $v_j \in [0, 1]$  that models the weight of  $p_j \in \mathcal{P}$ . Then (FP1) can be rewritten as

$$\begin{aligned}
 \text{(FP2)} \quad & \frac{1}{n} \sum_{i=1}^n u_i \leq \alpha \\
 & \left| \sum_{j:p_j \in R_i} v_j - s_i \right| \leq u_i \quad \text{for } i = 1 \dots n \tag{2} \\
 & \sum_{j=1}^m v_j \leq 1 \tag{3} \\
 & u_i, v_j \in [0, 1] \quad \text{for } i = 1 \dots n, j = 1 \dots m
 \end{aligned}$$

We require that the sum of weights of the discrete distribution we compute is at most 1 because, for any distribution returned with  $\sum_j^m v_j < 1$ , we can add an arbitrary point that is not contained in any range with weight  $1 - \sum_j^m v_j$ .

The above decision problem can be written as a linear program by replacing (2) with two linear inequalities.

$$\text{(LP1)} \quad -\frac{1}{n} \sum_{i=1}^n u_i \geq -\alpha \tag{4}$$

$$u_i - \sum_{j:p_j \in R_i} v_j \geq -s_i \quad \text{for } 1 \leq i \leq n \tag{5}$$

$$u_i + \sum_{j:p_j \in R_i} v_j \geq s_i \quad \text{for } 1 \leq i \leq n \tag{6}$$

$$\sum_{j=1}^m v_j \leq 1 \tag{7}$$

$$u_i, v_j \in [0, 1] \quad \text{for } i = 1 \dots n, j = 1 \dots m$$

It will be convenient to write the above LP in a compact form. Let  $\mathbf{u} = (u_1, \dots, u_n)$ ,  $\mathbf{v} = (v_1, \dots, v_m)$ , and  $\mathbf{x} = (\mathbf{u}, \mathbf{v})$ . Set

$$\mathbb{X} = \{\mathbf{x} = (\mathbf{u}, \mathbf{v}) \in [0, 1]^{n+m} \mid \|\mathbf{v}\|_1 \leq 1\}.$$

For  $0 \leq \kappa \leq 2n$ , let  $\mathbf{A}_\kappa \mathbf{x} \geq \mathbf{b}_\kappa$  denote the  $\kappa$ -th constraint of (4), (5), (6). Namely,  $\mathbf{A}_0 \mathbf{x} \geq \mathbf{b}_0$  denotes (4), and for  $1 \leq i \leq n$ ,  $\mathbf{A}_{2i-1} \mathbf{x} \geq \mathbf{b}_{2i-1}$  and  $\mathbf{A}_{2i} \mathbf{x} \geq \mathbf{b}_{2i}$  denote the constraints (5) and (6), respectively, for the rectangle  $R_i$ . Then (LP1) asks whether there exists an  $\mathbf{x} \in \mathbb{X}$  such that  $\mathbf{A} \mathbf{x} \geq \mathbf{b}$ .

We use a Multiplicative-Weight-Update (MWU) method to solve this linear program, following the general approach described by Arora et al. [5], though the exact implementation depends on the specific LP; see also [42]. We describe how this approach is implemented in our setting. We will need this algorithm for the faster implementation described in Section 3.

## 2.2 MWU algorithm

We describe an algorithm that either returns a  $\tilde{\mathbf{x}} \in \mathbb{X}$  such that  $\mathbf{A} \tilde{\mathbf{x}} \geq \mathbf{b} - \delta/4$  or returns that there is no feasible solution for (LP1). We set two parameters  $\eta = \frac{\delta}{c_1}$  and  $T = \lceil c_2 \delta^{-2} \ln(2n+1) \rceil$ , where  $c_1, c_2 > 0$  are sufficiently large constants to be chosen later. The algorithm works in  $T$  rounds. At the beginning of round  $t$ , it has a  $(2n+1)$ -dimensional probability vector  $\mathbf{w}^{(t)} = (w_0^{(t)}, \dots, w_{2n}^{(t)})$ . Initially,  $\mathbf{w}^{(1)} = (\frac{1}{2n+1}, \dots, \frac{1}{2n+1})$ . In the  $t$ -th round, the algorithm solves the decision problem consisting of one constraint

$$\mathbf{w}^{(t)\top} \mathbf{A} \mathbf{x} \geq \mathbf{w}^{(t)\top} \mathbf{b}, \quad \mathbf{x} \in \mathbb{X} \tag{8}$$

which we refer to as the *expected constraint*. The algorithm computes, as described below,  $\mathbf{x}^{(t)} = \arg \max_{\mathbf{x} \in \mathbb{X}} \mathbf{w}^{(t)\top} \mathbf{A} \mathbf{x}$ .

If  $\mathbf{w}^{(t)\top} \mathbf{A} \mathbf{x}^{(t)} < \mathbf{w}^{(t)\top} \mathbf{b}$ , we conclude that (LP1) is infeasible because, by definition, any feasible solution of (LP1) satisfies (8), and we return *NO*. Otherwise, for  $i = 0 \dots 2n$ , we set

$$w_i^{(t+1)} = w_i^{(t)} \frac{1 - \eta(\mathbf{A}_i \mathbf{x}^{(t)} - \mathbf{b}_i)}{\mu^{(t+1)}} \quad (9)$$

where  $\mu^{(t+1)}$  is a normalization factor so that  $\|\mathbf{w}^{(t+1)}\|_1 = 1$ .

If the algorithm does not return *NO* within the first  $T$  rounds, then after completing  $T$  rounds, it returns  $\bar{\mathbf{x}} = \frac{1}{T} \sum_{i=1}^T \mathbf{x}^{(i)}$ . Suppose  $\bar{\mathbf{x}} = (\bar{\mathbf{u}}, \bar{\mathbf{v}})$ . We return the distribution  $\tilde{\mathcal{D}} = \{(p_j, \tilde{v}_j) \mid \tilde{v}_j > 0\}$ .

**Computing  $\mathbf{x}^{(t)}$ .** We now describe the computation of  $\mathbf{x}^{(t)} = (u_1^{(t)}, \dots, u_n^{(t)}, v_1^{(t)}, \dots, v_m^{(t)})$  at each step. We can express the LHS of the expected constraint (8) as

$$\mathbf{w}^{(t)\top} \mathbf{A} \mathbf{x} = \sum_{i=1}^n \varphi_i^{(t)} u_i + \sum_{j=1}^m \psi_j^{(t)} v_j \quad (10)$$

$$\text{where } \varphi_i^{(t)} = w_{2i}^{(t)} + w_{2i-1}^{(t)} - \frac{w_0^{(t)}}{n} \quad \text{for } 1 \leq i \leq n, \quad (11)$$

$$\psi_j^{(t)} = \sum_{i: p_j \in R_i} w_{2i}^{(t)} - w_{2i-1}^{(t)} \quad \text{for } 1 \leq j \leq m. \quad (12)$$

Note that the two terms in (10) do not share variables and (7) does not involve  $u_i$ 's, so we can maximize each of them independently. Recall that  $u_i \in [0, 1]$ , so to maximize (10), we choose for  $1 \leq i \leq n$ ,

$$u_i^{(t)} = \begin{cases} 1 & \text{if } \varphi_i^{(t)} > 0, \\ 0 & \text{otherwise.} \end{cases} \quad (13)$$

Next, we choose  $v_j^{(t)}$  as follows. Let  $j^{(t)} = \arg \max_{1 \leq j \leq m} \psi_j^{(t)}$ . Since  $\mathbf{v} \in [0, 1]^m$  and  $\|\mathbf{v}\|_1 \leq 1$  for  $(\mathbf{u}, \mathbf{v}) \in \mathbb{X}$ , we choose for  $1 \leq j \leq m$ ,

$$v_j^{(t)} = \begin{cases} 1 & \text{if } j = j^{(t)} \text{ and } \psi_j^{(t)} > 0, \\ 0 & \text{otherwise.} \end{cases} \quad (14)$$

Note that although many  $u_i$ 's could be set to 1, at most one  $v_j$  is set to 1. This property will be crucial for our faster implementation in the next section. Since at most  $T$   $v_j$ 's are non-zero,  $|\tilde{\mathcal{D}}| \leq T = O(\delta^{-2} \log n)$ .

This completes the description of our basic algorithm. We now analyze its running time and correctness.

## 2.3 Analysis

► **Lemma 2.** *Assume that the MWU algorithm returns a solution  $\bar{\mathbf{x}}$  after  $T$  rounds. Then  $\mathbf{A}_i \bar{\mathbf{x}} \geq \mathbf{b}_i - \delta/4$ , for every  $0 \leq i \leq 2n$ .*

**Proof.** It is straightforward to verify that for any round  $1 \leq t \leq T$ ,  $|\mathbf{A}_i \mathbf{x}^{(t)} - \mathbf{b}_i| \leq 2$ . Let  $\mathcal{T}_i^- = \{t \leq T \mid \mathbf{A}_i \mathbf{x}^{(t)} - \mathbf{b}_i < 0\}$  be the subset of rounds where  $\mathbf{A}_i \mathbf{x}^{(t)} - \mathbf{b}_i < 0$ . Using the analysis in Arora et al. [5] (see the proof of Theorem 3.3), for every  $0 \leq i \leq 2n$ , we obtain,

$$\begin{aligned}
 0 &\leq \sum_{t=1}^T \frac{1}{2} (\mathbf{A}_i \mathbf{x}^{(t)} - \mathbf{b}_i) + \eta \sum_{t=1}^T \frac{1}{2} |\mathbf{A}_i \mathbf{x}^{(t)} - \mathbf{b}_i| + \frac{\ln(2n+1)}{\eta} \\
 &= (1+\eta) \sum_{t=1}^T \frac{1}{2} (\mathbf{A}_i \mathbf{x}^{(t)} - \mathbf{b}_i) + 2\eta \sum_{t \in \mathcal{T}_i^-} \frac{1}{2} |\mathbf{A}_i \mathbf{x}^{(t)} - \mathbf{b}_i| + \frac{\ln(2n+1)}{\eta} \\
 &\leq (1+\eta) \sum_{t=1}^T \frac{1}{2} (\mathbf{A}_i \mathbf{x}^{(t)} - \mathbf{b}_i) + 2\eta T + \frac{\ln(2n+1)}{\eta}.
 \end{aligned}$$

The last inequality follows because  $|\mathbf{A}_i \mathbf{x}^{(t)} - \mathbf{b}_i| \leq 2$ . Noting that  $\tilde{\mathbf{x}} = \frac{1}{T} \sum_{i=1}^T \mathbf{x}^{(t)}$ , we get  $0 \leq (1+\eta)(\mathbf{A}_i \tilde{\mathbf{x}} - \mathbf{b}_i) + 4\eta + \frac{2\ln(2n+1)}{\eta T}$ . By choosing  $c_1 = 32$  and  $c_2 = 512$ , we obtain  $\eta = \frac{\delta}{32}$  and  $T = \lceil 512\delta^{-2} \ln(2n+1) \rceil$ . Therefore

$$(1+\eta)(\mathbf{A}_i \tilde{\mathbf{x}} - \mathbf{b}_i) + \delta/4 \geq 0 \Leftrightarrow \mathbf{A}_i \tilde{\mathbf{x}} \geq \mathbf{b}_i - \delta/4. \quad \blacktriangleleft$$

► **Lemma 3.** *Given a training set  $\mathcal{Z}$  and parameters  $\alpha, \delta$ , the algorithm ISFEASIBLE either returns a discrete distribution  $\tilde{\mathcal{D}}$  of size  $O(\delta^{-2} \log n)$  such that  $\text{err}(\tilde{\mathcal{D}}, \mathcal{Z}) \leq \alpha + \delta/2$ , or returns NO for  $\alpha < \alpha^*$ .*

**Proof.** First, by definition, our algorithm always solves the expected constraint optimally. Hence, if the algorithm stops in iteration  $t \leq T$  (because we cannot satisfy the expected constraint  $\mathbf{w}^{(t)\top} \mathbf{A} \mathbf{x} \geq \mathbf{w}^{(t)\top} \mathbf{b}$  for  $\mathbf{x} \in \mathbb{X}$ ) then it is also true that there is no  $\mathbf{x} \in \mathbb{X}$  such that  $\mathbf{A} \mathbf{x} \geq \mathbf{b}$ . So the algorithm correctly returns that the LP is infeasible and  $\alpha < \alpha^*$ .

Next, assume that the algorithm returns  $\tilde{\mathcal{D}}$ . Recall that  $\tilde{\mathbf{x}} = (\tilde{\mathbf{u}}, \tilde{\mathbf{v}}) = (\tilde{u}_1, \dots, \tilde{u}_n, \tilde{v}_1, \dots, \tilde{v}_n) \in \mathbb{X}$ . Since  $\mathbb{X}$  is convex and  $\mathbf{x}^{(t)} \in \mathbb{X}$  for each  $1 \leq t \leq T$ , it also holds that  $\tilde{\mathbf{x}} \in \mathbb{X}$  and  $\|\tilde{\mathbf{v}}\|_1 \leq 1$ . Hence, it follows that  $\tilde{\mathcal{D}}$  is indeed a distribution of size at most  $T = O(\delta^{-2} \log n)$ . From Lemma 2 and the equivalence of (FP2) and (LP1), we get

$$\frac{1}{n} \sum_{i=1}^n \tilde{u}_i \leq \alpha + \delta/4 \quad \text{and} \quad \left| \sum_{j:p_j \in R_i} \tilde{v}_j - s_i \right| \leq u_i + \delta/4$$

for  $i = 1 \dots n$ . Recall that  $s_{\tilde{\mathcal{D}}}(R_i) = \sum_j \mathbb{1}(p_j \in R_i) \tilde{v}_j$ . Hence

$$\text{err}(\tilde{\mathcal{D}}, \mathcal{Z}) = \frac{1}{n} \sum_{i=1}^n |s_{\tilde{\mathcal{D}}}(R_i) - s_i| \leq \frac{1}{n} \sum_{i=1}^n \tilde{u}_i + \delta/4 \leq \alpha + \delta/4 + \delta/4 = \alpha + \delta/2. \quad \blacktriangleleft$$

As for the running time, the binary search executes  $O(\log \frac{1}{\delta})$  iterations, each of which runs the ISFEASIBLE algorithm. We need  $O(m \log n) = O(n^d \log n)$  time to construct (LP1). The ISFEASIBLE algorithm runs for  $T = O(\delta^{-2} \log n)$  rounds. In each round  $t$ , we need  $O(n)$  time to compute the values of  $\mathbf{u}^{(t)}$  by (11) and (13). We need  $O(m \log n) = O(n^d \log n)$  time to find all the values  $\psi_j^{(t)}$  by (12). Putting everything together, we have the following lemma.

► **Lemma 4.** *The algorithm runs in  $O(\delta^{-2} n^d \log^2 n \log \delta^{-1})$  time.*

### 3 The Improved Algorithm

We present an implementation of a small variant of the algorithm in the last section that computes the desired distribution with high probability in  $n(\delta^{-1} \log n)^{O(1)}$  time (see Theorem 9 below for a more precise characterization). There are two challenges in implementing

the ISFEASIBLE procedure efficiently. First, (LP1) has  $O(n^d)$  variables – although  $\mathbf{u}$  has dimension  $n$ ,  $\mathbf{v}$  has dimension  $m = O(n^d)$  – so we cannot afford to represent (LP1) explicitly. Second, in each round  $t$ , computing  $\mathbf{u}^{(t)}$  is easy, but computing  $\mathbf{v}^{(t)}$  quickly seems challenging even though only one of the values in  $\mathbf{v}$  is non-zero. We exploit underlying geometry to address both challenges. We first describe how to implement ISFEASIBLE without writing the LP explicitly and then describe how to compute  $\mathbf{x}^{(t)}$  quickly.

**Implicit representation of LP.** We maintain the probability vector  $\mathbf{w}^{(t)}$  as before. We also maintain a multi-set  $\mathcal{F} \subset \mathbb{R}^d$  of points. Initially,  $\mathcal{F} = \emptyset$ . Each round adds one point to  $\mathcal{F}$ , so  $|\mathcal{F}| \leq T = O(\delta^{-2} \log n)$ . Since we have  $\mathbf{w}^{(t)}$  at our disposal, we can compute  $u_i^{(t)}$ , for  $1 \leq i \leq n$ , using (11) and (13) as before. The main question is how to compute  $j^{(t)} = \arg \max_{1 \leq j \leq m} \psi_j^{(t)}$  without maintaining all the variables explicitly. For each rectangle  $R_i \in \mathcal{R}$ , let  $\omega^{(t)}(R_i) = w_{2i}^{(t)} - w_{2i-1}^{(t)}$ . For each point  $x \in \mathbb{R}^d$ , we define its *depth* with respect to  $(\mathcal{R}, \omega^{(t)})$ , denoted by  $\Delta(x)$ , to be

$$\Delta(x) = \sum_{R \in \mathcal{R}} \mathbb{1}(x \in R) \omega^{(t)}(R).$$

It is easily checked that the depth of all points lying in the same cell of  $\mathcal{A}(\mathcal{R})$  is the same, and that many cells may have the same depth. By the definition of the depth,  $\psi_j^{(t)} = \Delta(p_j)$ . Thus the problem of computing  $j^{(t)}$  is equivalent to choosing a point  $p^{(t)}$  in  $\mathbb{R}^d$  of the maximum depth, i.e.,  $\Delta(p^{(t)}) = \max_{p \in \mathbb{R}^d} \Delta(p)$ . Chan [9] has described an  $O(n^{d/2})$  time algorithm to compute a point of maximum depth in a set of  $n$  weighted rectangles in  $\mathbb{R}^d$ . We refer to this algorithm as DEEPESTPT( $\mathcal{R}, \omega$ ), where  $\mathcal{R}$  is a set of  $n$  rectangles in  $\mathbb{R}^d$  and  $\omega \in \mathbb{R}^n$  is the weight vector.

We thus proceed in the  $t$ -th round of ISFEASIBLE as follows. First, compute the vector  $\mathbf{u}^{(t)}$  as before. Next, we compute the deepest point  $p^{(t)}$  by calling DEEPESTPT( $\mathcal{R}, \omega^{(t)}$ ). If  $\Delta(p^{(t)}) > 0$ , then we add  $p^{(t)}$  to the set  $\mathcal{F}$ ; otherwise we ignore it. Intuitively, this is equivalent to setting  $v_{j^{(t)}} = 1$  if  $\psi_{j^{(t)}} > 0$  and 0 otherwise. We do not know how to compute  $j^{(t)}$  efficiently and thus cannot compute which  $v_j$  needs to be set to 1. But what comes to our rescue is that we only need to compute  $w_q^{(t)} \mathbf{A}_q \mathbf{x}$ , for all  $0 \leq q \leq 2n$ , to check whether the expected constraint holds and to update the weight factors. Fortunately, we can accomplish this using only  $\mathbf{u}$  and  $p^{(t)}$ , without an explicit representation of  $\mathbf{x}$ , as follows. We set  $w_{2i-1}^{(t)} \mathbf{A}_{2i-1} \mathbf{x} = w_{2i-1}^{(t)} u_i^{(t)}$  and  $w_{2i}^{(t)} \mathbf{A}_{2i} \mathbf{x} = w_{2i}^{(t)} u_i^{(t)}$  if  $\Delta(p^{(t)}) \leq 0$  or  $p^{(t)} \notin R_i$ , and we set  $w_{2i-1}^{(t)} \mathbf{A}_{2i-1} \mathbf{x} = w_{2i-1}^{(t)} (u_i^{(t)} - 1)$  and  $w_{2i}^{(t)} \mathbf{A}_{2i} \mathbf{x} = w_{2i}^{(t)} (u_i^{(t)} + 1)$  if  $\Delta(p^{(t)}) > 0$  and  $p^{(t)} \in R_i$ . It can be checked that these values are the same as when we set  $v_{j^{(t)}}$  as above.

After having computed  $w_q^{(t)} \mathbf{A}_q \mathbf{x}$  for all  $0 \leq q \leq 2n$ , we check whether  $\mathbf{w}^{(t)\top} \mathbf{A} \mathbf{x} < \mathbf{w}^{(t)\top} \mathbf{b}$ . If so, we stop and return NO. Otherwise, we compute  $\mathbf{w}^{(t+1)}$  using (9) as before. If the algorithm is not aborted within the first  $T$  rounds, we return  $\hat{\mathcal{D}} = \{(p, \frac{1}{T}) \mid p \in \mathcal{F}\}$ . Recall that  $\mathcal{F}$  is a multi-set. If there are  $s$  copies of a point  $p$ , we keep only one copy of  $p$  and set its weight to  $\frac{s}{T}$ . The following lemma establishes the correctness of the algorithm.

► **Lemma 5.** *Given  $\mathcal{Z}$  and  $\delta \in [0, 1]$ ,  $\text{err}(\tilde{\mathcal{D}}, \mathcal{Z}) = \text{err}(\hat{\mathcal{D}}, \mathcal{Z})$ .*

**Proof.** For simplicity, we assume that for any  $\omega^{(t)}$ ,  $1 \leq t \leq T$ , the maximum-depth cell in  $\mathcal{A}(\mathcal{R})$  with respect to  $\omega^{(t)}$  is unique and that  $j^{(t)}$  are distinct for each  $t \leq T$ . Then by the definition of depth,  $p^{(t)}$  and  $p_{j^{(t)}}$  lie in the same cell of  $\mathcal{A}(\mathcal{R})$ , so the value of  $\mathbf{w}^{(t)\top} \mathbf{A} \mathbf{x}$  computed by the implicit algorithm is the same as  $\mathbf{w}^{(t)\top} \mathbf{A} \mathbf{x}^{(t)}$  computed by the basic algorithm. Hence, assuming  $\mathbf{w}^{(t)}$  computed by the two algorithms is the same,  $\mathbf{w}^{(t+1)}$  computed by them is also the same. Recall that  $v_{j^{(t)}}^{(t)}$  is set to 1 if and only if  $\psi_{j^{(t)}}^{(t)} > 0$ , which is the same

## 18:10 Computing Data Distribution from Query Selectivities

condition when  $p^{(t)}$  is added to  $\mathcal{F}$  by the implicit algorithm.  $v_j^{(t)} = 1$  if and only if  $p^{(t)} \in F$ . Hence,  $p_j^{(t)} \in \text{supp}(\tilde{\mathcal{D}})$  if and only if  $p^{(t)} \in \text{supp}(\hat{\mathcal{D}})$  and  $p_j^{(t)}$  and  $p^{(t)}$  lie in the same cell of  $\mathcal{A}(\mathcal{R})$ . The same argument as in Lemma 1 now implies that  $\text{err}(\tilde{\mathcal{D}}, \mathcal{Z}) = \text{err}(\hat{\mathcal{D}}, \mathcal{Z})$ .  $\blacktriangleleft$

**Fast computation of  $p^{(t)}$ .** The main observation is that it is not crucial to compute  $x^{(t)} = \arg \max_{x \in \mathbb{X}} w^{(t)\top} \mathbf{A}x$  in the  $t$ -th round of the MWU algorithm. Instead, it suffices to compute  $\tilde{x}^{(t)}$  such that  $w^{(t)\top} \mathbf{A}\tilde{x}^{(t)} \geq w^{(t)\top} \mathbf{A}x^{(t)} - \delta/12$ . In the terminology of the implicit LP algorithm, this is equivalent to saying that it suffices to compute a point  $\tilde{p}^{(t)}$  with  $\Delta(\tilde{p}^{(t)}) \geq \max_{p \in \mathbb{R}^d} \Delta(p) - \delta/12$ . We use  $\varepsilon$ -approximations and random sampling [23, 11] to compute  $\tilde{p}^{(t)}$  quickly. The notion of  $\varepsilon$ -approximation is defined for general range spaces but we define  $\varepsilon$ -approximation in our setting.

Given a set  $\mathcal{R}$  of rectangles and a weight function  $\omega : \mathcal{R} \rightarrow [0, 1]$  a multi-subset  $\mathcal{N} \subset \mathcal{R}$  is called an  $\varepsilon$ -approximation of  $(\mathcal{R}, \omega)$  if for any point  $x \in \mathbb{R}^d$ ,

$$\left| \frac{\Delta(x, \mathcal{R}, \omega)}{\omega(\mathcal{R})} - \frac{\Delta(x, \mathcal{N})}{|\mathcal{N}|} \right| \leq \varepsilon, \quad (15)$$

where  $\Delta(x, \mathcal{R}, \omega)$  is the weighted depth of point  $x$  in the set of rectangles  $\mathcal{R}$  with weights  $\omega$ , and  $\Delta(x, \mathcal{N})$  is the depth of point  $x$  in the set of rectangles  $\mathcal{N}$  assuming that the weight of each rectangle in  $\mathcal{N}$  is 1.

Let  $A$  be a random (multi-)subset of  $\mathcal{R}$  of size  $O(\varepsilon^{-2} \ln \phi^{-1})$  where at each step each rectangle of  $\mathcal{R}$  is chosen with probability proportional to its weight (with repetition). It is well known [23, 11] that  $A$  is an  $\varepsilon$ -approximation with probability at least  $1 - \phi$ . With this result at our disposal, we compute  $\tilde{p}^{(t)}$ , an approximately deepest point with respect to  $(\mathcal{R}, w^{(t)})$ , as follows. Let  $w^{(t)}$  be as defined above. Let  $w^+ = (w_1^+, \dots, w_n^+)$  and  $w^- = (w_1^-, \dots, w_n^-)$  be two vectors such that  $w_i^+ = w_{2i}^{(t)}$  and  $w_i^- = w_{2i-1}^{(t)}$  for every  $1 \leq i \leq n$ . We set  $r = c_3 \delta^{-2}$ , where  $c_3 > 0$  is a sufficiently large constant. We repeat the following for  $\mu = O(\log n)$  times: For each  $h \leq \mu$ , we choose a random sample  $\mathcal{N}_h^+$  of  $(\mathcal{R}, w^+)$  of size  $r$  and another random sample  $\mathcal{N}_h^-$  of  $(\mathcal{R}, w^-)$  of size  $r$ . Let  $\mathcal{N}_h = \mathcal{N}_h^+ \cup \mathcal{N}_h^-$ . We define the weight function  $\bar{\omega}_h : \mathcal{N}_h \rightarrow \mathbb{R}$  as

$$\bar{\omega}_h(R) = \begin{cases} \frac{\|w^+\|_1}{r} & \text{if } R \in \mathcal{N}_h^+, \\ -\frac{\|w^-\|_1}{r} & \text{if } R \in \mathcal{N}_h^-. \end{cases} \quad (16)$$

We compute a deepest point  $\tilde{p}_h^{(t)}$  with respect to  $(\mathcal{N}_h, \bar{\omega}_h)$  along with  $\Delta(\tilde{p}_h^{(t)}, \mathcal{N}_h, \bar{\omega}_h)$  by calling `DEEPESTPT` $(\mathcal{N}_h, \bar{\omega}_h)$ . After repeating  $\mu$  times, we choose as  $\tilde{p}^{(t)}$  the point  $\tilde{p}_\xi^{(t)}$  with the median  $\Delta(\tilde{p}_\xi^{(t)}, \mathcal{N}_\xi, \bar{\omega}_\xi)$  among depths  $\{\Delta(\tilde{p}_1^{(t)}, \mathcal{N}_1, \bar{\omega}_1), \dots, \Delta(\tilde{p}_\mu^{(t)}, \mathcal{N}_\mu, \bar{\omega}_\mu)\}$ , where  $\xi \in [1, \mu]$ . Recall that  $\omega_i^{(t)} = w_{2i}^{(t)} - w_{2i-1}^{(t)}$ .

► **Lemma 6.**  $\Delta(\tilde{p}^{(t)}, \mathcal{R}, \omega^{(t)}) \geq \max_{x \in \mathbb{R}^d} \Delta(x, \mathcal{R}, \omega^{(t)}) - \delta/12$  with probability at least  $1 - 1/n^{O(1)}$ .

**Proof.** If we choose the constant  $c_3$  sufficiently large, then both  $\mathcal{N}_h^+$  and  $\mathcal{N}_h^-$  are  $\frac{\delta}{48}$ -approximations with probability greater than  $1/2$ . Therefore, for any point  $x \in \mathbb{R}^d$ ,

$$\left| \frac{\Delta(x, \mathcal{R}, w^+)}{\|w^+\|_1} - \frac{\Delta(x, \mathcal{N}_h^+)}{|\mathcal{N}_h^+|} \right|, \left| \frac{\Delta(x, \mathcal{R}, w^-)}{\|w^-\|_1} - \frac{\Delta(x, \mathcal{N}_h^-)}{|\mathcal{N}_h^-|} \right| \leq \frac{\delta}{48}. \quad (17)$$

Using (17) and the fact that  $\|w^+\|_1, \|w^-\|_1 \leq 1$ , we obtain

$$\begin{aligned} \Delta(x, \mathcal{R}, \omega^{(t)}) &= \Delta(x, \mathcal{R}, w^+) - \Delta(x, \mathcal{R}, w^-) \leq \frac{\|w^+\|_1}{r} \Delta(x, \mathcal{N}_h^+) - \frac{\|w^-\|_1}{r} \Delta(x, \mathcal{N}_h^-) + 2\frac{\delta}{48} \\ &= \Delta(x, \mathcal{N}_h, \bar{\omega}_h) + \frac{\delta}{24}. \end{aligned}$$

Similarly,  $\Delta(x, \mathcal{R}, \omega^{(t)}) \geq \Delta(x, \mathcal{N}_h, \bar{\omega}_h) - \frac{\delta}{24}$ . Next, we define  $x^* = \arg \max_{x \in \mathbb{R}^d} \Delta(x, \mathcal{R}, \omega^{(t)})$ . Hence, with probability greater than  $1/2$ ,

$$\Delta(\tilde{p}_h^{(t)}, \mathcal{R}, \omega^{(t)}) \geq \Delta(x^*, \mathcal{R}, \omega^{(t)}) - \frac{\delta}{12}.$$

Using the well known *median trick* (an application of the median trick can be found in [29]), we know that  $\tilde{p}^{(t)} = \tilde{p}_\xi^{(t)}$  that has the median depth satisfies

$$\Delta(\tilde{p}^{(t)}, \mathcal{R}, \omega^{(t)}) \geq \Delta(x^*, \mathcal{R}, \omega^{(t)}) - \frac{\delta}{12},$$

with probability at least  $1 - 1/n^{O(1)}$ .  $\blacktriangleleft$

► **Lemma 7.** *With probability at least  $1 - 1/n^{O(1)}$ , the ISFEASIBLE decision procedure either returns a discrete distribution  $\tilde{\mathcal{D}}$  of size  $O(\delta^{-2} \log n)$  such that  $\text{err}(\tilde{\mathcal{D}}, \mathcal{Z}) \leq \alpha + \delta/2$  or returns NO for  $\alpha < \alpha^*$ .*

**Proof.** Using the proofs of Lemma 2 and Lemma 3 (by slightly increasing the constants  $c_1, c_2$ ) we get that with probability at least  $1 - 1/n^{O(1)}$ , if the algorithm does not abort in the first  $T$  iterations in the end it finds  $\tilde{x} \in \mathbb{X}$  such that  $A\tilde{x} \geq \mathbf{b} - 3\delta/12 = \mathbf{b} - \delta/4$ . Using Lemma 5 and Lemma 3, we conclude the result.  $\blacktriangleleft$

► **Lemma 8.** *The improved algorithm runs in time*

$$O((n + \delta^{-2} \log^2 n + \delta^{-d} \log n) \delta^{-2} \log n \log \delta^{-1}).$$

**Proof.** In each round, we spend  $O(n)$  time to compute  $\mathbf{u}^{(t)}$ . For each  $h$ , we construct two  $\varepsilon$ -approximations  $\mathcal{N}_h^-, \mathcal{N}_h^+$  of size  $O(\delta^{-2})$  by applying weighted random sampling. Using a binary search tree constructed at the beginning of each round of the MWU algorithm, we get each sample in  $O(\log n)$  time. Hence, we construct  $\mathcal{N}_h$  in  $O(\delta^{-2} \log n)$ . We spend  $O((\delta^{-2})^{d/2})$  time to compute  $\tilde{p}_h^{(t)}$  using [9]. Overall we spend  $O((\delta^{-2} \log n + \delta^{-d}) \log n)$  time to compute the point  $\tilde{p}^{(t)}$ . Summing these bounds over all iterations of the binary search and the MWU method, we have the desired bound.  $\blacktriangleleft$

The algorithm we proposed computes a discrete distribution  $\tilde{\mathcal{D}}$  of size  $O(\delta^{-2} \log n)$ . The size can be reduced to  $O(\delta^{-2})$  as follows. We first run the algorithm above with  $\delta \leftarrow \delta/2$ . After obtaining  $\tilde{\mathcal{D}}$ , we repeat the following procedure  $O(\log n)$  times. In the  $h$ -th iteration, we get a  $\tilde{\mathcal{N}}_h$  of  $O(\delta^{-2})$  weighted random samples from  $\tilde{\mathcal{D}}$ . This is a  $\delta/2$ -approximation with probability at least  $1/2$ . Let  $\tilde{\mathcal{D}}_h$  be the distribution of size  $O(\delta^{-2})$  defined by  $\tilde{\mathcal{N}}_h$ . In the end, we return the best  $\delta/2$ -distribution  $\tilde{\mathcal{D}}_h$  with respect to  $\text{err}(\tilde{\mathcal{D}}_h, \mathcal{Z})$ . With probability at least  $1 - 1/n^{O(1)}$ , we find a distribution of size  $O(\delta^{-2})$  and additive error  $\delta$ . The running time of the additional steps is dominated by the running time in Lemma 8.

► **Theorem 9.** *Let  $\mathcal{Z} = \{z_1, \dots, z_n\}$  be a set of training samples such that  $z_i = (R_i, s_i)$ , where  $R_i$  is an axis aligned rectangle in  $\mathbb{R}^d$  and  $s_i \in [0, 1]$  is its selectivity. Given a parameter  $\delta \in (0, 1)$ , a discrete distribution  $\tilde{\mathcal{D}}$  of size  $O(\delta^{-2})$  can be computed in  $O((n + \delta^{-2} \log^2 n + \delta^{-d} \log n) \delta^{-2} \log n \log \delta^{-1})$  time such that  $\text{err}_1(\mathcal{Z}, \tilde{\mathcal{D}}) \leq \alpha_1^*(\mathcal{Z}) + \delta$ , with probability at least  $1 - 1/n^{O(1)}$ .*

► **Remark.** As we will see in the next section, our main algorithm can be extended to other settings. However, for axis-aligned rectangles, we can improve the dependency on  $d$  if we use a more sophisticated construction of  $\varepsilon$ -approximations for rectangular ranges. In fact, using [41] to construct the  $\varepsilon$ -approximation [11], we can get a distribution  $\tilde{\mathcal{D}}$  with the same properties as in Theorem 9 in  $O((n + \delta^{-6} \log^2 n + \delta^{-d/2}) \delta^{-2} \log n \log \delta^{-1})$  time.



## 4 Extensions

### 4.1 Extending to other error functions

So far, we only focused on the  $\ell_1$  empirical error. Our algorithm can be extended to  $\ell_\infty$  and  $\ell_2$  empirical errors with the same asymptotic complexity. In both cases, we run the same binary search as we had for the  $\ell_1$  error, and we call the `ISFEASIBLE`( $\mathcal{Z}, \delta, \alpha$ ).

**$\ell_\infty$  error.** Following the same arguments as in Section 2.1, we can formulate the problem as a simpler LP than (LP1). More specifically, by definition, we have to satisfy  $u_i \leq \alpha$  for  $1 \leq i \leq n$  instead of constraint (4). However, we observe that the linear problem is then equivalent to (LP1) setting  $u_i = \alpha$  for  $1 \leq i \leq n$ . Without having a vector  $\mathbf{u}$ , in each round  $t$ , the algorithm only computes the vector  $\mathbf{v}$  by computing  $p^{(t)}$  as we had in Section 3. The result follows.

**$\ell_2$  error.** The goal now is to find a distribution  $\mathcal{D}$  that minimizes  $\text{err}(\mathcal{D}, \mathcal{Z})$ . This is equivalent to the feasibility problem (LP1) replacing constraint (4) with

$$-\frac{1}{n} \sum_{i=1}^n u_i^2 \geq -\alpha. \quad (18)$$

Interestingly, the MWU method works even if the feasibility problem is on the form  $f(\mathbf{x}) \geq 0$ , where  $f(\cdot)$  is concave. It is straightforward to see that the new constraint (18) is concave. The rest of the constraints remain the same as in (LP1) so they are linear. Hence, we can use the same technique to optimize the new feasibility problem. Still the goal is to try and satisfy the expected constraint (8) by maximizing its LHS. In fact the new LHS of (8) is

$$\mathbf{w}^{(t)\top} \mathbf{A} \mathbf{x} = \sum_{i=1}^n -\frac{w_0^{(t)}}{n} u_i^2 + (w_{2i}^{(t)} + w_{2i-1}^{(t)}) u_i + \sum_{j=1}^m \psi_j^{(t)} v_j \quad (19)$$

The variables  $v_i$  are set by efficiently computing  $p^{(t)}$  as shown in Section 3. So we only focus on setting the variables  $u_i$ . Recall that our goal is to maximize the LHS in order to check if the expected constraint is satisfied. By simple algebraic calculations it is straightforward to see that the function  $g(u_i) = -\frac{w_0^{(t)}}{n} u_i^2 + (w_{2i}^{(t)} + w_{2i-1}^{(t)}) u_i$  under the constraint  $u_i \in [0, 1]$  is maximized for

$$u_i^{(t)} = \min \left\{ \frac{n(w_{2i}^{(t)} + w_{2i-1}^{(t)})}{2w_0^{(t)}}, 1 \right\}. \quad (20)$$

After computing  $\mathbf{w}^{(t)}$ , we can find  $\mathbf{u}^{(t)}$  in  $O(n)$  time. Hence, using (20) instead of (13) we can set  $\mathbf{u}$  that maximizes the expected constraint. All the other steps of the algorithm remain the same. Following the improved algorithm along with the proofs of Lemmas 2, 3, if  $\alpha > \alpha_2^*(\mathcal{Z})$ , with high probability, we get a distribution  $\tilde{\mathcal{D}}$  of size  $O(\delta^{-2})$  in near linear time such that  $\text{err}_2(\tilde{\mathcal{D}}, \mathcal{Z}) \leq \alpha + \delta/2$ .

► **Theorem 10.** *Let  $\mathcal{Z} = \{z_1, \dots, z_n\}$  be a set of training samples such that  $z_i = (R_i, s_i)$ , where  $R_i$  is an axis-aligned rectangle in  $\mathbb{R}^d$  and  $s_i \in [0, 1]$  is its selectivity. Given the parameters  $\delta \in (0, 1)$  and  $p \in \{1, 2, \infty\}$ , a discrete distribution  $\tilde{\mathcal{D}}$  of size  $O(\delta^{-2})$  can be computed in  $O((n + \delta^{-2} \log^2 n + \delta^{-d} \log n) \delta^{-2} \log n \log \delta^{-1})$  time such that  $\text{err}_p(\mathcal{Z}, \tilde{\mathcal{D}}) \leq \alpha_p^*(\mathcal{Z}) + \delta$  with probability at least  $1 - 1/n^{O(1)}$ .*

## 4.2 Extending to other types of ranges

Besides rectangles, our algorithm can be extended to more general ranges such as balls and halfspaces in  $\mathbb{R}^d$ . A *semi-algebraic* set in  $\mathbb{R}^d$  is a set defined by Boolean functions over polynomial inequalities, such as a unit semi-disc  $\{(x^2 + y^2 \leq 1) \cap (x \geq 0)\}$ , or an annulus  $\{(x^2 + y^2 \leq 4) \cap (x^2 + y^2 \geq 1)\}$ . Most familiar geometric shapes such as balls, halfspaces, simplices, ellipsoids, are semi-algebraic sets. The *complexity* of a semi-algebraic set is the sum of the number of polynomial inequalities and their maximum degree. See [6] for a discussion on semi-algebraic sets. Our results extend to semi-algebraic ranges of constant complexity. Let  $\mathcal{Z} = \{(\Gamma_1, s_1), \dots, (\Gamma_n, s_n)\}$  be a training set where each  $\Gamma_i$  is a semi-algebraic set of constant complexity. Let  $\Gamma = \{\Gamma_1, \dots, \Gamma_n\}$ . It is known that  $\mathcal{A}(\Gamma)$  has  $n^{O(d)}$  complexity, that for any weight function  $\omega: \Gamma \rightarrow [0, 1]$ , the deepest point can be computed in  $O(n^d)$  time, and that a random sample of size  $O(\delta^{-2} \log \phi^{-1})$  is a  $\delta$ -approximation with probability at least  $1 - \phi$ , [2, 3]. With these primitives at our disposal, the algorithm in Section 3 can be extended to this setting. Omitting all the details we obtain the following.

► **Theorem 11.** *Let  $\mathcal{Z} = \{z_1, \dots, z_n\}$  be a set of training samples such that  $z_i = (\Gamma_i, s_i)$ , where  $\Gamma_i$  is a semi-algebraic set of constant complexity and  $s_i \in [0, 1]$  is its selectivity. Given the parameters  $\delta \in (0, 1)$  and  $p \in \{1, 2, \infty\}$ , a discrete distribution  $\tilde{\mathcal{D}}$  of size  $O(\delta^{-2})$  can be computed in  $O((n + \delta^{-2} \log^2 n + \delta^{-2d} \log n) \delta^{-2} \log n \log \delta^{-1})$  time such that  $\text{err}_p(\mathcal{Z}, \tilde{\mathcal{D}}) \leq \alpha_p^*(\mathcal{Z}) + \delta$  with probability at least  $1 - 1/n^{O(1)}$ .*

## 5 Hardness Results

### 5.1 NP-Completeness

Let SELECTIVITY be the decision problem of constructing a distribution with small size and minimum error. More specifically, let  $\mathcal{Z} = \{(R_1, s_1), \dots, (R_n, s_n)\}$  be the input *training set* consisting of  $n$  rectangles  $R_1, \dots, R_n$  in  $\mathbb{R}^d$  along with their selectivities  $s_1, \dots, s_n$ , respectively. Let  $k$  be a positive natural number and  $\varepsilon \in [0, 1]$  be an error parameter. The SELECTIVITY problem asks if there exists a distribution  $\mathcal{D} \in \mathbb{D}_k$  such that  $\text{err}_p(\mathcal{D}, \mathcal{Z}) \leq \varepsilon$ . In the full version of the paper [1], we prove the following theorem.

► **Theorem 12.** *The SELECTIVITY problem is NP-Complete, even for  $d = 2$ .*

The NP-hardness proof is based on a gadget used by [37] to prove the NP-hardness of the *Square Cover* problem: given a set  $\mathcal{R}$  of  $n$  squares, and a parameter  $k$ , decide if there are  $k$  points  $S \in \mathbb{R}^2$  such that  $R \cap S \neq \emptyset$  for each  $R \in \mathcal{R}$ . We prove Theorem 12 by reducing 3-SAT to the SELECTIVITY problem in  $\mathbb{R}^2$ . Let  $(X, C)$  be the 3-SAT formula consisting of variables  $x_1, \dots, x_n$  and clauses  $C_1, \dots, C_m$ . We construct a set of weighted axis-aligned rectangles  $\mathcal{Z}$  and a number  $k$  such that  $(X, C)$  is satisfiable if and only if there exists a discrete distribution  $\mathcal{D}$  of size  $k$  such that  $\text{err}_p(\mathcal{D}, \mathcal{Z}) = 0$ .

In the above reduction, we construct a training set  $\mathcal{Z}$  such that there exists a  $\mathcal{D} \in \mathbb{D}_k$  with  $\text{err}_p(\mathcal{D}, \mathcal{Z}) = 0$  if and only if the formula is satisfiable. This construction implies a stronger result. Given  $\mathcal{Z}$  and  $k$ , let  $\alpha_{p,k}^* = \min_{\mathcal{D} \in \mathbb{D}_k} \text{err}_p(\mathcal{D}, \mathcal{Z})$ .

► **Corollary 13.** *Let  $f: \mathbb{N} \rightarrow \mathbb{N}$  be a monotonically non-decreasing function. Assuming  $P \neq NP$ , there is no polynomial-time algorithm that given a training set  $\mathcal{Z}$  and an input parameter  $k \in \mathbb{N}$  can compute  $\mathcal{D} \in \mathbb{D}_k$  such that  $\text{err}_p(\mathcal{D}, \mathcal{Z}) \leq f(n) \alpha_{p,k}^*(\mathcal{Z})$ .*

## 5.2 Conditional lower bounds

A natural question to ask is whether given  $\mathcal{Z}$  and  $k$ , a discrete distribution of size at most  $g(n)k$  can be computed in polynomial time such that the error is at most  $f(n)\alpha_{p,k}^*$ , where  $f(n), g(n)$  are monotonically non-decreasing non-negative functions. We prove conditional lower bounds for this problem, assuming the  $\text{FPT} \neq W[1]$  conjecture; see [14] for details of this conjecture.

Consider the following problem, which we refer to as **SELECTIVITY+**: given a training set  $\mathcal{Z}$  of  $n$  rectangles in  $\mathbb{R}^d$  along with their associated selectivities and a parameter  $k$ , compute a discrete distribution  $\mathcal{D} \in \mathbb{D}$  such that  $|\mathcal{D}| \leq g(n)k$  and  $\text{err}_p(\mathcal{D}, \mathcal{Z}) \leq f(n)\alpha_{p,k}^*$ , where  $g : \mathbb{N} \rightarrow \mathbb{N}$  and  $f(n) : \mathbb{N} \rightarrow \mathbb{N}$  are any functions that satisfy  $f(n) \geq 1$ ,  $g(n) \geq 1$ , and  $g(n) \cdot k = O(n)$ .

We prove a conditional lower bound on **SELECTIVITY+** by a reduction from the *coverage problem*: given a set  $\mathcal{R}$  of  $n$  rectangles in  $\mathbb{R}^d$  and a box  $B$ , does the union of rectangles in  $\mathcal{R}$  cover  $B$ , i.e.  $B \subseteq \bigcup_{R \in \mathcal{R}} R$ . The coverage problem is known to be  $W[1]$ -Hard, therefore, there cannot be an algorithm for coverage with running time  $h(d)n^{O(1)}$ , where  $h : \mathbb{N} \rightarrow \mathbb{N}$ , under the  $\text{FPT} \neq W[1]$  conjecture.

► **Theorem 14.** *The **SELECTIVITY+** problem cannot be solved in  $h(d)n^{O(1)}$  time, where  $h$  is any computable function  $h : \mathbb{N} \rightarrow \mathbb{N}$ , under the  $\text{FPT} \neq W[1]$  conjecture.*

**Proof.** Suppose on the contrary there is an algorithm  $\mathcal{A}$  for **SELECTIVITY+** that runs in  $h(d)n^{O(1)}$  time.

We show a reduction from the coverage problem. Let  $\mathcal{R}, B$ , as defined above, be an instance of the coverage problem. For each rectangle  $R_i \in \mathcal{R}$ , we construct  $z_i = (R_i, 0)$ , i.e., rectangle  $R_i$  with selectivity 0. Finally, we add  $z_n = (B, 1)$ . We set  $k = 1$ . The running time to construct the instance of the **SELECTIVITY+** problem is  $O(d \cdot n)$ . We then run  $\mathcal{A}$  on this instance.

We first consider the case when the input is a **NO** instance of the coverage problem. This implies that there exists a point  $q \in \mathbb{R}^d$  such that  $q \cap B \neq \emptyset$  and  $q \cap (\bigcup_{R_i \in \mathcal{R}} R_i) = \emptyset$ . Consider the distribution  $\mathcal{D} = \{(q, 1)\}$ . Notice that  $\text{err}_p(\mathcal{D}, \mathcal{Z}) = 0 = \alpha_{p,1}^* = \alpha_{p,k}^* = f(n)\alpha_{p,k}^*$  because  $q$  lies only inside  $B$ , where  $B$  has selectivity 1 and all other rectangles  $R_i$  have selectivity 0. By definition of **SELECTIVITY+**,  $\mathcal{A}$  in this instance must return a distribution  $\mathcal{D}$  such that  $\text{err}_p(\mathcal{D}, \mathcal{Z}) = 0$ .

In the **YES** instance of the coverage problem, if  $q \in B$  then  $q \in \bigcup_{R_i \in \mathcal{R}} R_i$ . Hence, if we add any point  $q \in B$  in distribution  $\mathcal{D}$  with positive probability then  $\text{err}_p(\mathcal{D}, \mathcal{Z}) > 0$  because the selectivity of every rectangle in  $\mathcal{R}$  is 0. On the other hand, the selectivity of  $B$  is 1 so if we do not add any point  $q \in B$  in  $\mathcal{D}$  with positive probability then  $\text{err}_p(\mathcal{D}, \mathcal{Z}) > 0$ . In any case,  $f(n)\alpha_{p,k}^* \geq \text{err}_p(\mathcal{D}, \mathcal{Z}) > 0$ . Actually, it is easy to verify that for any  $\mathcal{D} \in \mathbb{D}$ ,  $\text{err}_p(\mathcal{D}, \mathcal{Z}) > \frac{1}{2n^2}$  for any  $p \in \{1, 2, \infty\}$ . Thus,  $f(n)\alpha_{p,k}^* > \frac{1}{2n^2} > 0$ .

Overall, let  $\mathcal{D}$  be the distribution returned by  $\mathcal{A}$ . Given  $\mathcal{D}$ , we can compute  $\text{err}_p(\mathcal{D}, \mathcal{Z})$  in  $O(n^2)$  time, because  $|\mathcal{D}| \leq g(n)k \leq n$ . If  $\text{err}_p(\mathcal{D}, \mathcal{Z}) = 0$  then the solution to the coverage problem in the original instance is **NO**. Otherwise, if  $\text{err}_p(\mathcal{D}, \mathcal{Z}) > 0$ , then the solution to the coverage problem in the original instance is **YES**.

We thus obtain a  $h(d)n^{O(1)}$  time algorithm for the coverage problem, which is a contradiction under the  $\text{FPT} \neq W[1]$  conjecture. Hence,  $\mathcal{A}$  cannot run in  $h(d)n^{O(1)}$  time under the  $\text{FPT} \neq W[1]$  conjecture, as claimed. ◀

The above theorem implies there is no near-linear algorithm for SELECTIVITY+ under the  $\text{FPT} \neq W[1]$  conjecture. In fact, we can prove a stronger conditional lower bound. We define the APPROX-SELECTIVITY problem: given a training set  $\mathcal{Z}$  of  $n$  rectangles in  $\mathbb{R}^d$  along with their associated selectivities and a parameter  $\delta > 0$ , compute a  $\mathcal{D} \in \mathbb{D}$  such that  $|\mathcal{D}| \leq \frac{1}{\delta^3}$  and  $\text{err}_p(\mathcal{D}, \mathcal{Z}) \leq f(n)\alpha_{p,1}^* + \delta$ , where  $f(n) : \mathbb{N} \rightarrow \mathbb{N}$  is any function that satisfies  $f(n) \geq 1$ .

The requirements for APPROX-SELECTIVITY are considerably weak. First, we are allowed to use more points than our algorithm i.e.  $\frac{1}{\delta^3}$  points. Second, we only need to compete with a multiplicative factor of the best solution using just one point i.e.  $f(n)\alpha_{p,1}^*$ . It turns out that for small  $\delta$ , the conditions of the APPROX-SELECTIVITY become the same as SELECTIVITY+ and hence the proof technique for Theorem 14 also proves the following:

► **Theorem 15.** *The APPROX-SELECTIVITY problem cannot be solved in  $h(d)(n^{O(1)} + \delta^{-O(1)} + n^{O(1)}\delta^{-O(1)})$  time, where  $h$  is any computable function  $h : \mathbb{N} \rightarrow \mathbb{N}$ , under the  $\text{FPT} \neq W[1]$  conjecture.*

**Proof.** Similar to the proof of Theorem 14, we map an instance of the coverage problem to an instance of APPROX-SELECTIVITY, picking  $\delta = \frac{1}{8n^2}$ . The same argument shows that in the NO instance an algorithm for APPROX-SELECTIVITY returns a distribution with error at most  $\frac{1}{8n^2}$  while in the YES instance, it returns a distribution with error at least  $\frac{1}{2n^2}$ . ◀

► **Remark.** Notice that both theorems are based on reductions from the coverage problem. Chan [8] proved that the coverage problem is  $W[1]$ -hard with respect to dimension  $d$ , showing that the existence of a  $d$ -clique in a graph with  $\sqrt{n}$  vertices can be reduced to the coverage problem with  $O(n)$  boxes in  $\mathbb{R}^d$ . Thus, the time complexity of the  $d$ -clique problem is intimately related to our problem. If fast matrix multiplication is allowed, the best known algorithm for the  $d$ -clique problem requires  $\omega(n^{d/3})$  time [39].

Due to inefficiencies of fast matrix multiplication, there is a lot of interest in solving the clique problem using combinatorial algorithms (i.e. without fast matrix multiplication). However, the best current combinatorial algorithm for the  $d$ -clique problem in general graphs requires  $\Omega(n^d / \text{polylog}(n))$  running time. This lower bound implies that an  $o(n^{d/2-\eta})$  time combinatorial algorithm for the SELECTIVITY+ problem or a  $o(n + \delta^{-d/2+\eta})$  time combinatorial algorithm for the APPROX-SELECTIVITY problem, for any  $\eta > 0$ , is unlikely. Such an algorithm would lead to a  $o(n^{d-\eta})$  combinatorial algorithm for  $d$ -clique, solving a major open problem in graph theory.

## 6 Related Work

**Selectivity Estimation.** Selectivity estimation techniques can be broadly classified into three regimes: query-driven, data-driven, and hybrid. Most literature focuses on orthogonal range queries.

**Query-driven methods:** These methods derive selectivity estimates based on previous queries and their results. They do not require access to the underlying data distribution. Methods falling under this category include DQM [24] and the query-driven histogram techniques such as STHoles [7], Isomer [45], and QuickSel [40]. They construct models or histograms using results from previous queries and apply these models to estimate selectivity for new queries.

**Data-driven methods:** These methods, on the other hand, derive selectivity estimates from the underlying data distribution. They sample data and build statistical models that capture the data distribution, which are then used for selectivity estimation. There is long history along this direction, but some recent examples in this category include Naru [49], DQM-D [35], and DeepDB[26].

Hybrid methods: These methods take into account both the query workload and the underlying data distribution. They not only model the data distribution but also incorporate query feedback to refine the model over time. Examples of such methods include MSCN [28] and LW [15], which utilize both data and query features in a regression-based framework for selectivity estimation. See [47] for a detailed literature review.

Our method is query-driven, but is unique because none of query-driven and data-driven models above offered theoretical guarantees on the learnability of the functions or their learning procedures.

**Multiplicative-Weights-Update (MWU) method.** The MWU method has been independently rediscovered multiple times and has found applications in several areas including machine learning [17], game theory [18, 20], online algorithms [16], computational geometry [12], etc. In the context of optimization problems, there is a rich body of work on using MWU and related techniques to efficiently solve special classes of LPs and SDPs [30, 42, 50, 4]. The technique we use in this paper is an adaption of the one used by [42] to solve packing and covering LPs. See also [5] for a detailed overview. The MWU method has also been used to implicitly solve linear programs. A classical example is the multi-commodity flow problem where the describing the LP explicitly requires exponential number constraints [19]. Furthermore, there has been a lot of work related to implicitly solving special classes of LPs in computational geometry or combinatorial optimization using the MWU method [10, 12, 13], the primal dual method [48], or the ellipsoid method [22, 21].

Despite the geometric nature of our problem, to the best of our knowledge, all previous (geometric) techniques that implicitly solve an LP do not extend to our problem. In particular, in [12] the authors study various geometric packing and covering problems, such as the weighted set cover of points by disks, and the maximum weight independent set of disks, using the MWU method. The main difference from our setting is that for all problems they study, the matrix  $A$  and the vector  $b$  are non-negative. One of the main challenges in our setting is that we have to deal with both positive and negative values in  $A$  and  $b$ . It is not clear how their algorithms can be extended to our setting. For example, for the maximum weight independent set problem, they also describe an efficient algorithm to compute an approximately deepest point. However, their algorithm works in 2 dimensions while all ranges have positive weights. In our setting, we have ranges in any constant dimension  $d$  with both positive and negative weights.

## 7 Conclusion and future work

In this work, we studied the problem of finding the data distribution to fit a query training set consisting of axis-aligned rectangles (representing orthogonal range selectivity queries) with the smallest error. While the problem has been studied in the past, only an expensive  $\Omega(n^d)$  algorithm was known for constructing a distribution of size  $O(n^d)$ . We showed that the decision problem is NP-complete even for  $d = 2$ . Based on a standard complexity conjecture, we also gave conditional lower bounds showing that the exponential dependency on  $d$  is inevitable for additive or relative approximations. On the positive side, for the  $\ell_1$  empirical error, we gave a  $O((n + \delta^{-d})\delta^{-2} \text{polylog } n)$  time algorithm that returns a data distribution of size  $O(\delta^{-2})$  with additive error  $\delta$ . Furthermore, we showed that our algorithm for  $\ell_1$  error can be extended to  $\ell_2$  and  $\ell_\infty$ , as well as any type of ranges as long as they are algebraic sets of constant complexity. In view of our hardness results, significant improvements to our upper bounds are unlikely.

There are some interesting directions following from this work. What properties should the data and query distribution satisfy so that the running time is polynomial in  $d$  as well? In addition to selectivity queries, another interesting line of work is to study the construction of distributions for other types of database queries such as aggregation and joins.

---

## References

- 1 Pankaj Agarwal, Rahul Raychaudhury, Stavros Sintos, and Jun Yang. Computing data distribution from query selectivities. *CoRR*, abs/2401.06047, 2024. doi:10.48550/arXiv.2401.06047.
- 2 Pankaj K. Agarwal, Boris Aronov, Esther Ezra, and Joshua Zahl. Efficient algorithm for generalized polynomial partitioning and its applications. *SIAM Journal on Computing*, 50(2):760–787, 2021. doi:10.1137/19M1268550.
- 3 Pankaj K. Agarwal and Micha Sharir. Arrangements and their applications. In *Handbook of computational geometry*, pages 49–119. Elsevier, 2000. doi:10.1016/b978-044482537-7/50003-6.
- 4 Sanjeev Arora, Elad Hazan, and Satyen Kale. Fast algorithms for approximate semidefinite programming using the multiplicative weights update method. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*, pages 339–348. IEEE, 2005. doi:10.1109/SFCS.2005.35.
- 5 Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of computing*, 8(1):121–164, 2012. doi:10.4086/toc.2012.v008a006.
- 6 S. Basu, R. Pollack, and M. F. Roy. Algorithms in real algebraic geometry. In *Algorithms and Computation in Mathematics*. 2nd ed., Springer-Verlag, 2000.
- 7 Nicolas Bruno, Surajit Chaudhuri, and Luis Gravano. Stholes: A multidimensional workload-aware histogram. In Sharad Mehrotra and Timos K. Sellis, editors, *Proceedings of the 2001 ACM SIGMOD international conference on Management of data, Santa Barbara, CA, USA, May 21-24, 2001*, pages 211–222. ACM, 2001. doi:10.1145/375663.375686.
- 8 Timothy M. Chan. A (slightly) faster algorithm for klee’s measure problem. In *Proceedings of the twenty-fourth annual symposium on Computational geometry*, pages 94–100, 2008. doi:10.1145/1377676.1377693.
- 9 Timothy M. Chan. Klee’s measure problem made easy. In *2013 IEEE 54th annual symposium on foundations of computer science*, pages 410–419. IEEE, 2013. doi:10.1109/FOCS.2013.51.
- 10 Timothy M. Chan and Qizheng He. Faster approximation algorithms for geometric set cover. In *36th International Symposium on Computational Geometry (SoCG 2020)*, 2020. doi:10.4230/LIPIcs.SocG.2020.27.
- 11 Bernard Chazelle. *The discrepancy method: randomness and complexity*. Cambridge University Press, 2000.
- 12 Chandra Chekuri, Sarel Har-Peled, and Kent Quanrud. Fast lp-based approximations for geometric packing and covering problems. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1019–1038. SIAM, 2020. doi:10.1137/1.9781611975994.62.
- 13 Kenneth L. Clarkson and Kasturi Varadarajan. Improved approximation algorithms for geometric set cover. In *Proceedings of the twenty-first annual symposium on Computational geometry*, pages 135–141, 2005. doi:10.1145/1064092.1064115.
- 14 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 5(4). Springer, 2015. doi:10.1007/978-3-319-21275-3.
- 15 Anshuman Dutt, Chi Wang, Azade Nazi, Srikanth Kandula, Vivek R. Narasayya, and Surajit Chaudhuri. Selectivity estimation for range predicates using lightweight models. *Proc. VLDB Endow.*, 12(9):1044–1057, 2019. doi:10.14778/3329772.3329780.



- 16 Dean P. Foster and Rakesh Vohra. Regret in the on-line decision problem. *Games and Economic Behavior*, 29(1-2):7–35, 1999.
- 17 Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997. doi:10.1006/jcss.1997.1504.
- 18 Yoav Freund and Robert E. Schapire. Adaptive game playing using multiplicative weights. *Games and Economic Behavior*, 29(1-2):79–103, 1999.
- 19 Naveen Garg and Jochen Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM Journal on Computing*, 37(2):630–652, 2007. doi:10.1137/S0097539704446232.
- 20 Michael D. Grigoriadis and Leonid G. Khachiyan. A sublinear-time randomized approximation algorithm for matrix games. *Operations Research Letters*, 18(2):53–58, 1995. doi:10.1016/0167-6377(95)00032-0.
- 21 Martin Grötschel, László Lovász, and Alexander Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1:169–197, 1981. doi:10.1007/BF02579273.
- 22 Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric algorithms and combinatorial optimization*, volume 2. Springer Science & Business Media, 2012.
- 23 Sarel Har-Peled. Geometric approximation algorithms. In *Mathematical Surveys and Monographs*, volume 173. American Mathematical Soc., 2011.
- 24 Shohedul Hasan, Saravanan Thirumuruganathan, Jeess Augustine, Nick Koudas, and Gautam Das. Deep learning models for selectivity estimation of multi-attribute queries. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 1035–1050, 2020. doi:10.1145/3318464.3389741.
- 25 David Haussler. Decision theoretic generalizations of the pac model for neural net and other learning applications. *Information and computation*, 100(1):78–150, 1992. doi:10.1016/0890-5401(92)90010-D.
- 26 Benjamin Hilprecht, Andreas Schmidt, Moritz Kulessa, Alejandro Molina, Kristian Kersting, and Carsten Binnig. Deepdb: learn from data, not from queries! *Proceedings of the VLDB Endowment*, 13(7):992–1005, 2020. doi:10.14778/3384345.3384349.
- 27 Xiao Hu, Yuxi Liu, Haibo Xiu, Pankaj K. Agarwal, Debmalya Panigrahi, Sudeepa Roy, and Jun Yang. Selectivity functions of range queries are learnable. In *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12–17, 2022*, pages 959–972. ACM, 2022. doi:10.1145/3514221.3517896.
- 28 Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter A. Boncz, and Alfons Kemper. Learned cardinalities: Estimating correlated joins with deep learning. In *9th Biennial Conference on Innovative Data Systems Research, CIDR 2019, Asilomar, CA, USA, January 13–16, 2019, Online Proceedings*, 2019. URL: <http://cidrdb.org/cidr2019/papers/p101-kipf-cidr19.pdf>.
- 29 Alexander Kogler and Patrick Traxler. Parallel and robust empirical risk minimization via the median trick. In *Mathematical Aspects of Computer and Information Sciences: 7th International Conference, MACIS 2017, Vienna, Austria, November 15–17, 2017, Proceedings*, pages 378–391. Springer, 2017. doi:10.1007/978-3-319-72453-9\_31.
- 30 Christos Koufogiannakis and Neal E. Young. Beating simplex for fractional packing and covering linear programs. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007), October 20–23, 2007, Providence, RI, USA, Proceedings*, pages 494–504. IEEE Computer Society, 2007. doi:10.1109/FOCS.2007.16.
- 31 Richard J. Lipton, Jeffrey F. Naughton, and Donovan A. Schneider. Practical selectivity estimation through adaptive sampling. In *Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, pages 1–11, 1990. doi:10.1145/93597.93611.



- 32 Ryan Marcus, Parimarjan Negi, Hongzi Mao, Nesime Tatbul, Mohammad Alizadeh, and Tim Kraska. Bao: Making learned query optimization practical. *SIGMOD Rec.*, 51(1):6–13, 2022. doi:10.1145/3542700.3542703.
- 33 Ryan Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, and Nesime Tatbul. Neo: a learned query optimizer. *Proceedings of the VLDB Endowment*, 12(11), 2019. doi:10.14778/3342263.3342644.
- 34 Ryan Marcus and Olga Papaemmanouil. Deep reinforcement learning for join order enumeration. In *Proceedings of the First International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*, pages 1–4, 2018. doi:10.1145/3211954.3211957.
- 35 Volker Markl, Peter J. Haas, Marcel Kutsch, Nimrod Megiddo, Utkarsh Srivastava, and Tam Minh Tran. Consistent selectivity estimation via maximum entropy. *The VLDB journal*, 16:55–76, 2007. doi:10.1007/s00778-006-0030-1.
- 36 Yossi Matias, Jeffrey Scott Vitter, and Min Wang. Wavelet-based histograms for selectivity estimation. In *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 448–459, 1998. doi:10.1145/276304.276344.
- 37 Nimrod Megiddo and Kenneth J. Supowit. On the complexity of some common geometric location problems. *SIAM J. Comput.*, 13(1):182–196, 1984. doi:10.1137/0213014.
- 38 Parimarjan Negi, Ryan Marcus, Hongzi Mao, Nesime Tatbul, Tim Kraska, and Mohammad Alizadeh. Cost-guided cardinality estimation: Focus where it matters. In *2020 IEEE 36th International Conference on Data Engineering Workshops (ICDEW)*, pages 154–157. IEEE, 2020. doi:10.1109/ICDEW49219.2020.00034.
- 39 Jaroslav Nešetřil and Svatopluk Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 26(2):415–419, 1985.
- 40 Yongjoo Park, Shucheng Zhong, and Barzan Mozafari. Quicksel: Quick selectivity learning with mixture models. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 1017–1033, 2020. doi:10.1145/3318464.3389727.
- 41 Jeff M. Phillips. Algorithms for epsilon-approximations of terrains. In *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part I: Tack A: Algorithms, Automata, Complexity, and Games*, volume 5125 of *Lecture Notes in Computer Science*, pages 447–458. Springer, 2008. doi:10.1007/978-3-540-70575-8\_37.
- 42 Serge A. Plotkin, David B. Shmoys, and Éva Tardos. Fast approximation algorithms for fractional packing and covering problems. *Mathematics of Operations Research*, 20(2):257–301, 1995. doi:10.1287/moor.20.2.257.
- 43 Viswanath Poosala, Peter J. Haas, Yannis E. Ioannidis, and Eugene J. Shekita. Improved histograms for selectivity estimation of range predicates. *ACM Sigmod Record*, 25(2):294–305, 1996. doi:10.1145/233269.233342.
- 44 Viswanath Poosala and Yannis E. Ioannidis. Selectivity estimation without the attribute value independence assumption. In *VLDB*, volume 97, pages 486–495, 1997. URL: <http://www.vldb.org/conf/1997/P486.PDF>.
- 45 Utkarsh Srivastava, Peter J. Haas, Volker Markl, Marcel Kutsch, and Tam Minh Tran. ISOMER: consistent histogram construction using query feedback. In Ling Liu, Andreas Reuter, Kyu-Young Whang, and Jianjun Zhang, editors, *Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006, 3-8 April 2006, Atlanta, GA, USA*, page 39. IEEE Computer Society, 2006. doi:10.1109/ICDE.2006.84.
- 46 Markus Stocker, Andy Seaborne, Abraham Bernstein, Christoph Kiefer, and Dave Reynolds. Sparql basic graph pattern optimization using selectivity estimation. In *Proceedings of the 17th international conference on World Wide Web*, pages 595–604, 2008. doi:10.1145/1367497.1367578.
- 47 Xiaoying Wang, Changbo Qu, Weiyuan Wu, Jiannan Wang, and Qingqing Zhou. Are we ready for learned cardinality estimation? *Proceedings of the VLDB Endowment*, 14(9):1640–1654, 2021. doi:10.14778/3461535.3461552.

## 18:20 Computing Data Distribution from Query Selectivities

- 48 David P. Williamson. The primal-dual method for approximation algorithms. *Mathematical Programming*, 91:447–478, 2002. doi:10.1007/s101070100262.
- 49 Zongheng Yang, Eric Liang, Amog Kamsetty, Chenggang Wu, Yan Duan, Xi Chen, Pieter Abbeel, Joseph M. Hellerstein, Sanjay Krishnan, and Ion Stoica. Deep unsupervised cardinality estimation. *Proceedings of the VLDB Endowment*, 13(3):279–292, 2019. doi:10.14778/3368289.3368294.
- 50 Neal E. Young. Sequential and parallel algorithms for mixed packing and covering. In *Proceedings 42nd IEEE symposium on foundations of computer science*, pages 538–546. IEEE, 2001. doi:10.1109/SFCS.2001.959930.

# Information Inequality Problem over Set Functions

Miika Hannula  

University of Helsinki, Finland

---

## Abstract

Information inequalities appear in many database applications such as query output size bounds, query containment, and implication between data dependencies. Recently Khamis et al. [14] proposed to study the algorithmic aspects of information inequalities, including the information inequality problem: decide whether a linear inequality over entropies of random variables is valid. While the decidability of this problem is a major open question, applications often involve only inequalities that adhere to specific syntactic forms linked to useful semantic invariance properties. This paper studies the information inequality problem in different syntactic and semantic scenarios that arise from database applications. Focusing on the boundary between tractability and intractability, we show that the information inequality problem is  $\text{coNP}$ -complete if restricted to normal polymatroids, and in polynomial time if relaxed to monotone functions. We also examine syntactic restrictions related to query output size bounds, and provide an alternative proof, through monotone functions, for the polynomial-time computability of the entropic bound over simple sets of degree constraints.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Database theory

**Keywords and phrases** entropy, information theory, worst-case output size, computational complexity

**Digital Object Identifier** 10.4230/LIPIcs.ICDT.2024.19

**Related Version** *Full Version:* <https://arxiv.org/abs/2309.11818> [9]

**Funding** *Miika Hannula:* The author has been supported by the ERC grant 101020762.

## 1 Introduction

Information inequalities are linear constraints on entropies of random variables. Often referred to as the laws of information, these inequalities describe what is not possible in information theory. More than three decades ago, Pippenger asked whether all such laws follow from the *polymatroidal axioms* [25], depicted in Fig. 1. The polymatroidal axioms are also known to be equivalent to the non-negativity of Shannon’s information measures, which consist of entropy, conditional entropy, mutual information, and conditional mutual information. The inequality constraints derivable from the polymatroidal axioms are hence called *Shannon* inequalities. Pippenger’s question was famously answered in the negative by Zhang and Yeung who were the first to find a non-Shannon information inequality that is valid over entropies [28]. Zhang and Yeung’s proof was based on a novel innovation, identified as the *copy lemma* in [4], which still today remains essentially the only tool to establish novel non-Shannon inequalities [8].

Constraints on entropies are known to have many applications in database theory. Lee [19, 20] observed already in the 80s that database constraints can alternatively be expressed as equalities over information measures. More recently, the implication problem for data dependencies has been connected to validity of information inequalities [13], information theory has been used to analyze normal forms in relational and XML data models [1], and query containment for conjunctive queries under bag semantics – a notoriously difficult problem to study – has been proven to be equivalent in certain special cases to checking information inequalities involving maximum [15]. Perhaps the most fruitful application has been the use of information inequalities to obtain tight output size bounds for database queries [2, 6, 7, 11, 16, 17], and the subsequent development of *worst-case optimal join algorithms* that run in time proportional to these bounds [16, 17, 23, 24].



© Miika Hannula;

licensed under Creative Commons License CC-BY 4.0

27th International Conference on Database Theory (ICDT 2024).

Editors: Graham Cormode and Michael Shekelyan; Article No. 19; pp. 19:1–19:20

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Recently Khamis et al. [14] initiated the study of the algorithmic properties of information inequalities. The most central problem, called the information inequality problem, is to decide whether a given information inequality is valid over all entropic functions. The decidability of this problem is a major open question in the foundations of information theory. It was shown in [14] that checking the validity of monotone Boolean combinations of information inequalities (including the aforementioned *max-inequalities*) is co-recursively enumerable (co-r.e.). Since the implication problem for conditional independence implication is undecidable [18, 21], validity for general Boolean combinations of information inequalities is known to be undecidable. While the focus of [14] was on generalizations of the information inequality problem, this paper shifts attention to simplifications of the problem. Many applications, such as implication problems or query output size bounds, are related to information inequalities that adhere to specific syntactic forms. These syntactic forms are also often linked to semantic invariance properties which render the associated problems computable and sometimes even tractable. Identifying factors that make the information inequality problem either easy or hard is thus a task that can prove beneficial in multiple application scenarios.

This paper examines the information inequality problem with respect to different syntactic restrictions and semantic settings, focusing in particular on the boundary between tractable and intractable cases. We demonstrate that different factors, including the influence of the coefficients and the expressiveness of the information measures, give rise to coNP-completeness with respect to normal polymatroids (the subset of entropic functions associated with a non-negative I-measure [13, 27]), and disagreement between normal polymatroids and entropic functions. Our findings also reveal that when we relax the semantics to monotone functions or restrict it to modular functions (an implicit result in existing literature), the information inequality problem can be solved in polynomial time. Additionally, we demonstrate that this problem becomes polynomial-time solvable when we impose syntactic restrictions linked to cases where computing the entropic query output size bound is known to be in polynomial time. Finally, we identify a syntactic restriction over which monotone and entropic functions agree, leading to an alternative proof for the previously established fact [11] that the entropic bound is polynomial-time computable over simple sets of degree constraints.

## 2 Preliminaries

We write  $[n]$  for the set of integers  $\{1, \dots, n\}$ . We usually use boldface letters to denote sets. For two sets  $\mathbf{X}$  and  $\mathbf{Y}$ , we write  $\mathbf{XY}$  to denote their union. If  $A$  is an individual element, we sometimes write  $A$  instead of  $\{A\}$  to denote the singleton set consisting of  $A$ .

### 2.1 Relational databases

Fix disjoint countably infinite sets  $\text{Var}$  and  $\text{Val}$  of variables and values. Each variable  $A \in \text{Var}$  is associated with a subset of  $\text{Val}$ , called the *domain* of  $A$ , denoted  $\text{Dom}(A)$ . For a vector  $\mathbf{X} = (A_1, \dots, A_n)$  of variables, we write  $\text{Dom}(\mathbf{X})$  for the Cartesian product  $\text{Dom}(A_1) \times \dots \times \text{Dom}(A_n)$ . Given a finite set of variables  $\mathbf{X}$ , an  $\mathbf{X}$ -tuple is a mapping  $t : \mathbf{X} \rightarrow \text{Val}$  such that  $t(A) \in \text{Dom}(A)$ . We write  $\text{Tup}(\mathbf{X})$  for the set of all  $\mathbf{X}$ -tuples. For  $\mathbf{Y} \subseteq \mathbf{X}$ , the *projection*  $t[\mathbf{Y}]$  of  $t$  on  $\mathbf{Y}$  is the unique  $\mathbf{Y}$ -tuple that agrees with  $t$  on  $\mathbf{X}$ . A *relation*  $R$  over  $\mathbf{X}$  is a subset of  $\text{Tup}(\mathbf{X})$ . The variable set  $\mathbf{X}$  is also called (*relation*) *schema* of  $R$ . We sometimes write  $R(\mathbf{X})$  instead of  $R$  to emphasize that  $\mathbf{X}$  is the schema of  $R$ . For  $\mathbf{Y} \subseteq \mathbf{X}$ , the *projection* of  $R$  on  $\mathbf{Y}$ , written  $R[\mathbf{Y}]$ , is the set of all projections  $t[\mathbf{Y}]$  where  $t \in R$ . A *database* is a finite collection of relations  $D = \{R_1^D(\mathbf{X}_1), \dots, R_n^D(\mathbf{X}_n)\}$ . We assume in this paper that each relation is finite.

## 2.2 Information theory

Let  $X$  be a random variable associated with a finite domain  $D = \text{Dom}(X)$  and a probability distribution  $p : D \rightarrow [0, 1]$ , where  $\sum_{a \in D} p(a) = 1$ . The *entropy* of  $X$  is defined as

$$H(X) := - \sum_{x \in D} p(x) \log p(x). \quad (1)$$

Entropy is non-negative and does not exceed the logarithm of the domain size:  $0 \leq H(X) \leq \log |D|$ . In particular,  $H(X) = 0$  if and only if  $X$  is constant (i.e.,  $p(a) = 1$  for some  $a \in D$ ), and  $H(X) = \log |D|$  if and only if  $X$  is uniformly distributed (i.e.,  $p(a) = 1/|D|$  for all  $a \in D$ ).

Fix  $n \geq 1$  and consider a set of variables  $\mathbf{X} = \{X_1, \dots, X_n\}$ . We will use  $\alpha$  for subsets of  $[n]$ , and write  $\mathbf{X}_\alpha := \{X_i \mid i \in \alpha\}$ . In the following, we list some common classes of vectors  $\mathbf{h} = (h_\alpha)_{\alpha \subseteq [n]} \in \mathbb{R}^{2^n}$ . Note that such vectors  $\mathbf{h}$  can alternatively be conceived as functions from  $\mathcal{P}(\mathbf{X})$  to  $\mathbb{R}$ , called *set functions*. Hence we often write  $h(\mathbf{X}_\alpha)$  to denote the element  $h_\alpha$  of  $\mathbf{h}$ , and from now on refer to  $\mathbf{h}$  as a *function*. We assume  $h(\emptyset) = 0$  for all functions  $\mathbf{h}$ . For a list of functions  $\mathbf{h}_1, \dots, \mathbf{h}_n$ , the function  $c_1 \mathbf{h}_1 + \dots + c_n \mathbf{h}_n$  is called a *positive combination* (resp. *non-negative combination*) of  $\mathbf{h}_1, \dots, \mathbf{h}_n$  if  $c_i > 0$  (resp.  $c_i \geq 0$ ) for all  $i \in [n]$ .

**Polymatroids.** If  $\mathbf{h}$  satisfies the *polymatroidal axioms* (Fig. 1), it is called a *polymatroid*. The set of polymatroids over  $n$  is denoted  $\Gamma_n$ .

1.  $h(\emptyset) = 0$
2.  $h(\mathbf{X} \cup \mathbf{Y}) \geq h(\mathbf{X})$  (monotonicity)
3.  $h(\mathbf{X}) + h(\mathbf{Y}) \geq h(\mathbf{X} \cap \mathbf{Y}) + h(\mathbf{X} \cup \mathbf{Y})$  (submodularity)

■ **Figure 1** Polymatroidal axioms.

**Monotone functions.** If  $\mathbf{h}$  satisfies the first two axioms of the polymatroidal axioms, we call it a *monotone function*, and denote the set of monotone functions over  $n$  by  $\text{Mon}_n$ .

**Entropic functions.** Consider a relation  $R$  over a set  $\mathbf{X} = \{X_i\}_{i=1}^n$  of variables with finite domains, associated with a probability distribution  $p : R \rightarrow [0, 1]$ . Each subset  $\mathbf{Y} \subseteq \mathbf{X}$  can be viewed as a random variable with domain  $D = R[\mathbf{Y}]$  and probability distribution  $p_{\mathbf{Y}}(t) = \sum_{t' \in R, t'[\mathbf{Y}] = t} p(t')$ . In particular, the subset  $\mathbf{Y}$  is thus associated with an entropy  $H(\mathbf{Y})$ . The function  $\mathbf{h} = (H(\mathbf{X}_\alpha))_{\alpha \subseteq [n]}$  arising from  $p$  in this way is called an *entropic function*. Each entropic function is a polymatroid, but in the converse direction there are polymatroids which are not entropic functions. In general entropic functions satisfy many additional constraints which do not follow by the polymatroidal axioms alone. However, it is not known whether there exists any effective procedure to check that a given function is entropic. The *entropic region*  $\Gamma_n^* \subseteq \mathbb{R}^{2^n}$  consists of all entropic functions over  $n$ . The *almost entropic region*  $\bar{\Gamma}_n^*$  is defined as the topological closure of  $\Gamma_n^*$ .

**Normal polymatroids and step functions.** For  $U \subseteq \mathbf{X}$ , the function

$$s_U(\mathbf{W}) = \begin{cases} 0 & \text{if } \mathbf{W} \subseteq U; \\ 1 & \text{otherwise;} \end{cases} \quad (2)$$

## 19:4 Information Inequality Problem over Set Functions

is called a *step function*. We also use the notation  $s^{\mathbf{V}}$  to denote the step function  $s_{\mathbf{X} \setminus \mathbf{V}}$ . Note that the step function  $s^{\mathbf{V}}$  is the entropic function arising from the uniform distribution of two tuples  $t$  and  $t'$  such that  $t(A) \neq t'(A)$  if and only if  $A \in \mathbf{V}$ . The set of all step functions over  $n$  is denoted  $\mathbf{S}_n$ . A *normal polymatroid* is a positive combination of step functions, and the set of all normal polymatroids over  $n$  is denoted  $\mathbf{N}_n$ .

**Modular functions.** A polymatroid  $\mathbf{h}$  is called *modular* if the submodularity inequality (Fig. 1) is an equality:  $h(\mathbf{X}) + h(\mathbf{Y}) = h(\mathbf{X} \cap \mathbf{Y}) + h(\mathbf{X} \cup \mathbf{Y})$ . Alternatively, a function  $\mathbf{h}$  is modular if it is non-negative and such that  $h(\mathbf{X}) = \sum_{Y \in \mathbf{X}} h(Y)$ . The set of modular functions over  $n$  is denoted  $\text{Mod}_n$ . Modular functions can alternatively be defined in terms of *basic modular functions*, which are step functions  $s^{\{A\}}$  defined in terms of singleton sets  $\{A\}$ . We denote the set of all basic modular functions over  $n$  by  $\mathbf{B}_n$ . A function  $\mathbf{f}$  is a modular function if and only if it is a positive combination of basic modular functions.

By continuity of Eq. (1), and since there are no restrictions on domain sizes,  $c\mathbf{h}$  is entropic for  $c > 0$  and step functions  $\mathbf{h}$ . Furthermore, if  $\mathbf{h}$  and  $\mathbf{h}'$  are entropic functions defined by probability distributions  $p$  and  $p'$  over some relation  $R$ , the distribution  $p''(t \otimes t') := p(t)p'(t')$  on the direct product  $\{t \otimes t' \mid t, t' \in R\}$ ,  $(t \otimes t')(X) := (t(X), t'(X))$ , defines  $\mathbf{h} + \mathbf{h}'$ . This shows that the entropic region is closed under multiplication by positive integers, even though in general it is not closed under positive scalar multiplication [27]; in other words,  $\Gamma_n^*$  is not a *cone*. We conclude that both modular and normal polymatroids are entropic. In fact, Kenig and Suciú [13] have shown that the normal polymatroids are exactly those entropic functions that have a non-negative I-measure [27]. The introduced set functions are related to one another in the following way:

$$\begin{array}{ccccccc} \mathbf{B}_n & \subseteq & \mathbf{S}_n & & & & \\ \cap & & \cap & & & & \\ \text{Mod}_n & \subseteq & \mathbf{N}_n & \subseteq & \Gamma_n^* & \subseteq & \overline{\Gamma_n^*} \subseteq \Gamma_n \subseteq \text{Mon}_n. \end{array}$$

If  $n \geq 4$ , then all the above subset relations are strict.

We will repeatedly refer to the following Shannon's information measures (over some  $\mathbf{h}$ ).

- Conditional entropy:  $h(\mathbf{Y} \mid \mathbf{X}) := h(\mathbf{XY}) - h(\mathbf{X})$ .
- Mutual information:  $I_{\mathbf{h}}(\mathbf{X}; \mathbf{Y}) := h(\mathbf{X}) + h(\mathbf{Y}) - h(\mathbf{XY})$ .
- Conditional mutual information:  $I_{\mathbf{h}}(\mathbf{Y}; \mathbf{Z} \mid \mathbf{X}) := h(\mathbf{XY}) + h(\mathbf{XZ}) - h(\mathbf{X}) - h(\mathbf{XYZ})$ .

We may drop the subscript  $\mathbf{h}$  if it is clear from the context.

An *information inequality* is an expression  $\phi$  of the form

$$c_1 h(\mathbf{X}_1) + \cdots + c_k h(\mathbf{X}_k) \geq 0, \quad (3)$$

where  $c_i \in \mathbb{R}$ , and  $\mathbf{X}_i$  are sets of variables from  $\{X_j\}_{j=1}^n$ . We sometimes write  $\phi(\mathbf{X})$  instead of  $\phi$  to emphasize that the set of variables appearing in  $\phi$  is  $\mathbf{X}$ . For  $V \subseteq \mathbb{R}^{2^n}$ , we say that  $\phi$  is *valid* over  $V$ , denoted  $V \models \phi$ , if it holds true for all functions  $\mathbf{h} \in V$ .

► **Example 1.** Suppose  $X$  and  $Y$  are independent and uniformly either 0 or 1, and let  $Z = X + Y \pmod{2}$ . This joint distribution can be constructed by taking the uniform distribution over the relation  $R$  in Tab. 1. Let  $\mathbf{h}$  be the entropic function arising from this distribution. Let  $\phi$  be an information inequality of the form  $I_{\mathbf{h}}(X, Y, Z) \geq 0$  where

$$I_{\mathbf{h}}(X, Y, Z) := h(XYZ) - h(XY) - h(XZ) - h(YZ) + h(X) + h(Y) + h(Z) \quad (4)$$

is the mutual information of variables  $X, Y, Z$ . We observe that  $\phi$  is not true for  $\mathbf{h}$  because Eq. (4) evaluates to  $-1$ . In particular, this means that  $\Gamma_3^* \not\models \phi$ . On the other hand,  $\phi$  is true if we interpret  $\mathbf{h}$  as any step function  $s_{\mathbf{U}}$ ,  $\mathbf{U} \subseteq \{X, Y, Z\}$ . Since normal polymatroids are positive combinations of step functions, this entails  $\mathbf{N}_3 \models \phi$ .

■ **Table 1** The relation  $R$  representing  $X + Y \equiv Z \pmod{2}$ .

R	X	Y	Z
	0	0	0
	0	1	1
	1	0	1
	1	1	0

This paper focuses on the *information inequality problem* (IIP), introduced in [14], which is to decide whether a given information inequality is valid over  $\Gamma_n^*$ . This problem is co-r.e. [14], as the continuity of the entropy (1) and the density of the rationals in the reals imply that enumeration of all rational distributions will eventually lead to a counterexample of (3), if one exists at all. We introduce the following relativized version of IIP. Fixing sets of functions  $F_n \subseteq \mathbb{R}^{2^n}$ ,  $n \geq 1$ , and a set  $\mathcal{C}$  of information inequalities, the *information inequality problem over  $F_n$  w.r.t.  $\mathcal{C}$*  ( $\text{IIP}_{F_n}(\mathcal{C})$ ) is to determine whether a given information inequality  $\phi \in \mathcal{C}$  over  $n$  variables is valid over  $F_n$ . We leave out  $F_n$  (resp.  $\mathcal{C}$ ) if  $F_n = \Gamma_n^*$  (resp.  $\mathcal{C}$  contains all information inequalities). Note that an inequality  $\phi$  is valid over the entropic region  $\Gamma_n^*$  if and only if it is valid over the almost entropic region  $\overline{\Gamma_n^*}$ . To see why,  $V \models \phi$  is tantamount to  $V \subseteq C_\phi$ , where  $C_\phi = \{\mathbf{h} \in \mathbb{R}^{2^n} \mid \mathbf{h} \models \phi\}$ , and by taking closures on both sides,  $\Gamma_n^* \subseteq C_\phi$  entails  $\overline{\Gamma_n^*} \subseteq C_\phi$ . More generally, validity over  $\Gamma_n^*$  and  $\overline{\Gamma_n^*}$  disagrees with respect to Boolean combinations of information inequalities [12, 14]. Since our focus is on the information inequality problem alone, we now drop the almost entropic region  $\overline{\Gamma_n^*}$  from discussions.

Before proceeding, we shortly discuss input representation. We assume that the coefficients are rational. Note that in [14] the inputs of IIP and other related problems are vectors  $\mathbf{c} \in \mathbb{Z}^{2^n}$  representing the coefficients in Eq. (3). In this paper, we consider the input as a sequence  $((c_1, \mathbf{X}_1), \dots, (c_k, \mathbf{X}_k))$ , which is potentially exponentially shorter than the aforementioned coefficient vector  $\mathbf{c}$ . This distinction is not important if one is solely interested in decidability, as is the case in [14]. Since our aim is to chart the tractability boundary for different information inequality problems, we opt for the latter more concise representation. Furthermore, we assume that the coefficients themselves are encoded in binary.

We begin our analysis from intractable examples, and then move on to discuss tractable cases and their connections to query output bounds.

### 3 Intractable cases

Kenig and Suciu [13] establish an interesting connection between information inequalities and the implication problem for database dependencies. Fix a relation schema  $\mathbf{X}$  of  $n$  variables. An expression of the form  $\sigma = (\mathbf{V}; \mathbf{W} \mid \mathbf{U})$  is called a *conditional independence* (CI). If  $UVW = \mathbf{X}$ ,  $\sigma$  is specifically called a *saturated conditional independence* (SCI), and if  $\mathbf{V} = \mathbf{W}$ , it is called a *conditional* and shortened as  $(\mathbf{V} \mid \mathbf{U})$ . Lee [19] observed that an SCI of the form  $(\mathbf{V}; \mathbf{W} \mid \mathbf{U})$  holds true on the uniform distribution of a database relation  $R(UVW)$  if and only if  $R$  satisfies the corresponding *multivalued dependency* (MVD)  $\mathbf{U} \twoheadrightarrow \mathbf{V}$ . An analogous correspondence can be drawn between a conditional  $(\mathbf{V} \mid \mathbf{U})$  and the *functional dependency* (FD)  $\mathbf{U} \rightarrow \mathbf{V}$ . The results in [13] entail that if  $\Sigma$  is a set of SCIs and conditionals, and  $\tau$  is a conditional, then for any  $V$  such that  $N_n \subseteq V \subseteq \Gamma_n$ ,

$$V \models \sum_{\sigma \in \Sigma} h(\sigma) \geq h(\tau) \iff \Sigma \models \tau, \quad (5)$$



## 19:6 Information Inequality Problem over Set Functions

where the right-hand side denotes implication between the corresponding MVDs and FDs over database relations. Whether or not  $\sum_{\sigma \in \Sigma} h(\sigma) \geq h(\tau)$  is valid over  $V$  can be thus decided in polynomial time, because the implication problem for MVDs and FDs is known to be in polynomial time [3].

There are at least two ways to make the inequality in Eq. (5) harder. One possibility is to allow more complex information measures, much like how one can allow more expressive database dependencies in the implication problem. For instance, once the aforementioned syntactic restrictions are lifted, the implication problem for CIs becomes undecidable in both database theory (where CIs are known as embedded multivalued dependencies) and probability theory [10, 18, 21]. Another possibility, which does not seem to have a counterpart in the implication problem, is to permit coefficients distinct from 1. Next, we consider both of these strategies in isolation, considering first complex information measures.

The mutual information of two random variables generalizes to the multivariate mutual information over a set of random variables  $\mathbf{S}$ . For a general set function  $\mathbf{h}$ , the *multivariate mutual information* is given as

$$I_{\mathbf{h}}(\mathbf{S}) = \sum_{\mathbf{T} \subseteq \mathbf{S}} (-1)^{|\mathbf{T}|-1} \mathbf{h}(\mathbf{T}). \quad (6)$$

Again, we drop the subscript whenever this is possible without confusion. A particular case of the multivariate mutual information is the three-variate one, presented in Eq. (4). Multivariate mutual information is non-negative on step functions, but, as discussed in Example 1, it can be negative on entropic functions. Next we show that solving inequalities containing three-variate mutual informations and conditional entropies can already be coNP-hard, even if each coefficient is exactly one. The result, proven by a reduction from monotone satisfiability, holds for step functions but does not extend to entropic functions.

A conjunctive normal form Boolean formula  $\phi$  is called *monotone* if each clause in  $\phi$  contains only negative or only positive literals. The *monotone satisfiability problem* is the problem of deciding whether such a formula  $\phi$  has a satisfying truth assignment. This problem is well known to be NP-complete [5], and it remains NP-complete even if each clause consists of exactly three distinct literals [22]. Let us denote this restriction of the problem by 3DMONSAT. An instance of 3DMONSAT can be represented as a pair  $\phi = (\phi^+, \phi^-)$ , where  $\phi^+$  (resp.  $\phi^-$ ) is the set of all positive (resp. negative) clauses of  $\phi$ , and each clause is a set of exactly 3 variables.

► **Theorem 2.** *The information inequality problem over normal polymatroids is coNP-complete.*

**Proof.** Since normal polymatroids are positive combinations of step functions, and inequalities are preserved under positive combinations, the information inequality problems over step functions and normal polymatroids coincide. The upper bound is thus obvious. For the lower bound, we present a reduction from the complement of 3DMONSAT to the information inequality problem over step functions. Let  $\phi = (\phi^+, \phi^-)$  be an instance of 3DMONSAT. Suppose  $\mathbf{X}$  is the set of variables appearing in  $\phi$ . We may assume without loss of generality that every satisfying assignment must map at least one variable to 1.

Define an information inequality

$$\sum_{\mathbf{C} \in \phi^+} h(\mathbf{X} | \mathbf{C}) + \sum_{\mathbf{C} \in \phi^-} I(\mathbf{C}) \geq h(\mathbf{X}), \quad (7)$$

where  $I(\mathbf{C})$  is the three-variate mutual information (4) over the variables of  $\mathbf{C}$ , and  $h(\mathbf{X} | \mathbf{C})$  is the conditional entropy of  $\mathbf{X}$  given  $\mathbf{C}$ .

Each subset  $\mathbf{Y} \subseteq \mathbf{X}$  determines a unique step function  $s^{\mathbf{Y}}$  (Eq. 2) and a unique Boolean assignment

$$m_{\mathbf{Y}}(A) = \begin{cases} 1 & \text{if } A \in \mathbf{Y}, \\ 0 & \text{otherwise.} \end{cases}$$

We claim that  $m_{\mathbf{Y}}$  satisfies  $\phi$  if and only if Eq. (7) is false for  $h = s^{\mathbf{Y}}$ .

Assume first that  $m_{\mathbf{Y}}$  satisfies  $\phi$ . By our assumption some variable is mapped to 1, which means that  $\mathbf{Y}$  is non-empty. In particular,  $s^{\mathbf{Y}}(\mathbf{X}) = 1$ . For any positive clause  $\mathbf{C} \in \phi^+$ , we have  $\mathbf{C} \cap \mathbf{Y} \neq \emptyset$ , and consequently  $s^{\mathbf{Y}}(\mathbf{X} \mid \mathbf{C}) = s^{\mathbf{Y}}(\mathbf{X}) - s^{\mathbf{Y}}(\mathbf{C}) = 0$ . For any negative clause  $\mathbf{C} \in \phi^-$ , we have  $\mathbf{C} \not\subseteq \mathbf{Y}$ , in which case it is straightforward to verify that  $I(\mathbf{C}) = 0$ . We conclude that Eq. (7) is false for  $h = s^{\mathbf{Y}}$ .

Assume then that  $m_{\mathbf{Y}}$  does not satisfy  $\phi$ . If  $s^{\mathbf{Y}}(\mathbf{X}) = 0$ , then Eq. (7) is trivially true for  $h = s^{\mathbf{Y}}$  by the non-negativity of the conditional entropy and the multivariate mutual information on step functions. Suppose then  $s^{\mathbf{Y}}(\mathbf{X}) \neq 0$ , in which case  $s^{\mathbf{Y}}(\mathbf{X}) = 1$ . Assuming  $m_{\mathbf{Y}}$  does not satisfy some  $\mathbf{C} \in \phi^+$ , we have  $\mathbf{C} \cap \mathbf{Y} = \emptyset$  implying  $s^{\mathbf{Y}}(\mathbf{X} \mid \mathbf{C}) = 0$ . Assuming  $m_{\mathbf{Y}}$  does not satisfy some  $\mathbf{C} \in \phi^-$ , we have  $\mathbf{C} \subseteq \mathbf{Y}$  implying  $I(\mathbf{C}) = 1$ . We conclude that, for  $h = s^{\mathbf{Y}}$ , the left-hand side of Eq. (7) is at least 1, and thus the inequality holds. This concludes the proof of the claim.

The claim implies that  $\phi$  is not satisfiable if and only if Eq. (7) is valid over step functions. The theorem statement follows, since the reduction is clearly polynomial.  $\blacktriangleleft$

Observe that Eq. (7) is syntactically similar to the inequality in Eq. (5) in that each coefficient is exactly one. The difference comes from allowing three-variate mutual information, whereas the inequality in Eq. (5) allows only specific forms of conditional mutual information. The above proof moreover establishes *strong* coNP-completeness, because the problem remains coNP-complete even under unary encoding of the coefficients.

Alternatively, the preceding theorem can be proven by reducing 3-colorability to inequalities that allow the coefficients to grow while using only conditionals. Let  $G = (\mathbf{V}, \mathbf{E})$  be a graph consisting of a vertex set  $\mathbf{V}$  and a set of undirected edges  $\mathbf{E}$ . For each node  $A \in \mathbf{V}$ , introduce variables  $A_r, A_g, A_b$  representing possible colors of  $A$ . Assume that the graph contains  $n$  vertices. We define

$$\sum_{\substack{c \in \{r,g,b\} \\ A \in \mathbf{V}}} h(A_c) + \sum_{\substack{c,d \in \{r,g,b\} \\ c \neq d \\ A \in \mathbf{V}}} (2n+1)h(\mathbf{V} \mid A_c A_d) + \sum_{\substack{c \in \{r,g,b\} \\ \{A,B\} \in \mathbf{E}}} (2n+1)h(\mathbf{V} \mid A_c B_c) \geq (2n+1)h(\mathbf{V}). \quad (8)$$

Then  $G$  is three-colorable if and only if Eq. (8) is not valid over step functions (Appendix A). This way of proving Theorem 2 establishes also strong coNP-completeness, since each coefficient is bounded by a polynomial in the input size. It is necessary to allow coefficients other than 1 in Eq. (8). Otherwise, the equivalence (5) holds, meaning that the validity problem is equivalent to the implication problem for FDs, which is in polynomial time.

► **Example 3.** Eq. (8) behaves differently for step functions and entropic functions, even though both functions are non-negative on all the occurring information measures; in contrast, the proof of Theorem 2 relied on three-variate mutual information which is only guaranteed to be non-negative for step functions but can be negative for entropic functions. For a concrete example, suppose  $G$  is the complete graph of four vertices  $A, B, C, D$ . Since  $G$  is not three-colorable, Eq. (8) is valid over step functions. For the entropic function arising from the uniform distribution of Tab. 2, however, Eq. (8) is false.

■ **Table 2** Three-tuple counterexample.

$A_r$	$B_g$	$C_b$	$A_g$	$B_b$	$C_r$	$A_b$	$B_r$	$C_g$	$D_r$	$D_g$	$D_b$
0	0	0	0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	0	0	0	1	1	1
1	1	1	0	0	0	0	0	0	2	2	2

## 4 Tractable cases

We have seen that intractable inequalities arise from (i) complex information measures even if coefficients are restricted to 1, (ii) more simple information measures if coefficients are allowed to grow, and (iii) inequalities where the negative coefficients are associated with sets of size at most two. In this section we consider restrictions that give rise to inequalities solvable in polynomial time. We are specifically interested in inequalities of the form  $\sum_{\sigma \in \Sigma} w_i h(\sigma) \geq h(\mathbf{X})$  where  $w_i \geq 0$ , and  $\Sigma$  is a set of conditionals. Such inequalities make appearance when information theory is applied to obtain tight upper bounds for query output sizes. Since inequalities of the form (8) are intractable, imposing syntactic restrictions on  $\Sigma$  becomes necessary.

We next introduce information-theoretic query upper bounds, after which we move on to discuss the complexity of related syntactic restrictions and semantic modifications of the information inequality problem.

### 4.1 Query upper bounds

Fix a relation  $R$  over a variable set  $\mathbf{X}$ . Given vectors  $\mathbf{U}, \mathbf{V}$  of variables from  $\mathbf{X}$ , and values  $\mathbf{u} \in \text{Dom}(\mathbf{U})$ , the  $V$ -degree of  $\mathbf{U} = \mathbf{u}$  in  $R$ , denoted  $\text{deg}_R(\mathbf{V} \mid \mathbf{U} = \mathbf{u})$ , is the number of distinct values of  $\mathbf{V}$  that occur in  $R$  together with the value  $\mathbf{u}$  of  $\mathbf{U}$ . The max- $\mathbf{V}$ -degree of  $\mathbf{U}$ , denoted  $\text{deg}_R(\mathbf{V} \mid \mathbf{U})$  is the maximum  $V$ -degree of  $\mathbf{U} = \mathbf{u}$  over all  $\mathbf{u}$ . Expressions of the form  $\text{deg}_R(\sigma) \leq B$  (omitting the parentheses of  $\sigma$ ), where  $\sigma$  is a conditional and  $B \geq 1$ , are usually called *degree constraints*. Note that  $\text{deg}_R(\mathbf{V} \mid \mathbf{U}) = 1$  if and only if  $R$  satisfies the functional dependency  $\mathbf{U} \rightarrow \mathbf{V}$ . A  $\Sigma$ -inequality is an information inequality  $\phi_\Sigma(\mathbf{X}, \mathbf{w})$  of the form

$$\sum_{\sigma \in \Sigma} w_\sigma h(\sigma) \geq h(\mathbf{X}), \quad (9)$$

where  $\mathbf{w} = (w_\sigma)_{\sigma \in \Sigma}$  is a sequence of non-negative reals.

Fix a *self-join-free full conjunctive query*, i.e., a quantifier-free first-order formula of the form

$$Q(\mathbf{X}) = R_1(\mathbf{X}_1) \wedge \cdots \wedge R_n(\mathbf{X}_n),$$

where  $R_i(\mathbf{X}_i)$  are *relational atoms* over distinct *relation names*  $R_i$ , and variable sequences  $\mathbf{X}_i$  such that  $\mathbf{X}$  lists all the variables occurring in them. Note that this incurs a slight abuse of notation, because  $R_i(\mathbf{X}_i)$  could also refer to a relation  $R_i$  over  $\mathbf{X}_i$ . We also blur the distinction between a set and a sequence of variables  $\mathbf{X}_i$ , and say that a set of conditionals  $\Sigma$  is *guarded* by  $Q$  if every  $\sigma = (\mathbf{V} \mid \mathbf{U})$  from  $\Sigma$  is associated with a relation name  $R_i$ , called the *guard* of  $\sigma$  and denoted  $R_\sigma$ , such that  $\mathbf{U}\mathbf{V} \subseteq \mathbf{X}_i$ . A sequence of the form  $\mathbf{B} = (B_\sigma)_{\sigma \in \Sigma}$ ,  $B_\sigma \geq 1$ , form the *degree values* associated with  $\Sigma$ . A database  $D$  containing relations  $R_\sigma$ ,  $\sigma \in \Sigma$ , *satisfies* a conditionals-values pair  $(\Sigma, \mathbf{B})$ , written  $D \models (\Sigma, \mathbf{B})$ , if  $\text{deg}_{R_\sigma}(\sigma) \leq B_\sigma$  for all  $\sigma \in \Sigma$ .

For a set  $S \subseteq \mathbb{R}^{2^n}$ , and a set of conditionals  $\Sigma$  guarded by  $Q(\mathbf{X})$  and associated with values  $\mathbf{B}$ , define the *bound* of  $Q$  w.r.t.  $\Sigma, S, \mathbf{B}$  as

$$\text{Bound}_S(Q, \Sigma, \mathbf{B}) := \inf_{\substack{\mathbf{w} \geq 0 \\ S \models \phi_\Sigma(\mathbf{X}, \mathbf{w})}} \prod_{\sigma \in \Sigma} B_\sigma^{w_\sigma}.$$

The bounds  $\text{Bound}_{\text{Mod}_n}, \text{Bound}_{\text{N}_n}, \text{Bound}_{\Gamma_n^*}, \text{Bound}_{\Gamma_n}$  are often referred to as the *modular bound*, the *coverage bound*, the *entropic bound*, and the *polymatroid bound*. Writing  $Q(D)$  for the output of  $Q$  on a database  $D = \{R_1^D(\mathbf{X}_1), \dots, R_n^D(\mathbf{X}_n)\}$ , one can prove that the entropic bound is valid:  $|Q(D)| \leq \text{Bound}_{\Gamma_n^*}(Q, \Sigma, \mathbf{B})$  whenever  $D \models (\Sigma, \mathbf{B})$ . Since  $\Gamma_n^* \subseteq \Gamma_n$ , the entropic bound is less than or equal to the polymatroid bound. The entropic bound is asymptotically tight, but it is open whether or not the bound is computable. The polymatroid bound can be attained by solving a linear program of exponential size, but it is not tight. For a derivation of the entropic bound and a discussion on the asymptotic tightness (or lack thereof) of these bounds, we refer the reader to [26].

Fortunately, there are well-behaving syntactic restrictions for sets of conditionals  $\Sigma$ , some of which are presented next.

- For  $\sigma = (\mathbf{V} \mid \mathbf{U})$ , where  $\mathbf{U} = \emptyset$ , degree constraints of the form  $\deg_R(\sigma) \leq B$  are called *cardinality constraints*. The AGM bound [2] can be viewed as the entropic bound over a specific set of cardinality constraints.
- $\Sigma$  is called *acyclic* if the following directed graph is acyclic: the vertices are the variables in  $\mathbf{X}$ , and there is an edge from  $A$  to  $B$  if  $A \in \mathbf{X}$  and  $B \in \mathbf{Y} \setminus \mathbf{X}$ , for some  $(\mathbf{Y} \mid \mathbf{X}) \in \Sigma$ .
- $\Sigma$  is called *simple* if  $|\mathbf{U}| \leq 1$  for each  $(\mathbf{V} \mid \mathbf{U}) \in \Sigma$ .

The entropic bound is polynomial-time computable in all of these cases. Let us call the  $\Sigma$ -inequality (9) *acyclic* (resp. *simple*) if the underlying set  $\Sigma$  is acyclic (resp. simple). The sets of conditionals underlying cardinality constraints are vacuously acyclic, and validity for acyclic  $\Sigma$ -inequalities coincides for modular functions, entropic functions, and polymatroids [23]. Consequently, the entropic bound becomes computable in polynomial time through a linear program describing the validity of Eq. (9) over basic modular functions. Validity for simple  $\Sigma$ -inequalities similarly coincides for entropic functions, polymatroids, and normal polymatroids. This does not immediately entail that the entropic bound for simple  $\Sigma$  is computable in polynomial time, because normal polymatroids are constructed with step functions, and there are exponentially many step functions in the number of variables. The entropic bound is nevertheless known to be polynomial-time computable in this case, as has been shown recently [11].

Eq. (8) can now be viewed as an  $\Sigma$ -inequality (9) (up to scaling) arising from  $\Sigma$  that does not belong to any of the aforementioned well-behaving classes. Since validity of inequalities of the form Eq. (8) is **coNP**-hard over step functions (Appendix A), this immediately gives us the following result.

► **Theorem 4.** *The information inequality problem over normal polymatroids w.r.t.  $\Sigma$ -inequalities is **coNP**-complete. This problem remains **coNP**-hard even if  $|\mathbf{U}| \leq 2$  for all  $(\mathbf{V} \mid \mathbf{U}) \in \Sigma$ .*

Related to the previous result, computing the coverage bound over a set of conditionals is known to be **NP**-hard, and computing the polymatroid bound over an arbitrary set of conditionals can be efficiently reduced to computing the polymatroid bound over another set of conditionals  $(\mathbf{V} \mid \mathbf{U})$  such that  $|\mathbf{U}| \leq 2$  and  $|\mathbf{V}| \leq 3$  [11].

We now turn to discuss tractable cases of the information inequality problem obtained either by syntactic restrictions or semantic modifications. The syntactic restrictions we consider correspond quite closely to the aforementioned acyclic/simple  $\Sigma$ -inequalities.

## 4.2 Modular functions

Since modular functions can be constructed as positive combinations of basic modular functions, the information inequality problem is trivially polynomial-time computable in this context.

► **Proposition 5.** *The information inequality problem over modular functions is in polynomial time.*

One example of a syntactic class with respect to which validity over entropic functions corresponds to validity over modular functions are the acyclic  $\Sigma$ -inequalities. Given an acyclic set  $\Sigma$  of conditionals  $(\mathbf{V} \mid \mathbf{U})$ , and a polymatroid  $\mathbf{h}$  over  $\mathbf{X}$ , one can construct a modular function  $\mathbf{f}$  such that (i)  $f(\mathbf{X}) = h(\mathbf{X})$ , and (ii)  $f(\mathbf{V} \mid \mathbf{U}) \leq h(\mathbf{V} \mid \mathbf{U})$  for all  $(\mathbf{V} \mid \mathbf{U}) \in \Sigma$  [23]. Consequently, validity of acyclic  $\Sigma$ -inequalities (9) coincides for polymatroids, entropic functions, normal polymatroids, and modular functions. Thus it is known that all the aforementioned bounds (modular, coverage, entropic, and polymatroid bounds) coincide and are polynomial-time computable if  $\Sigma$  is acyclic. With respect to the information inequality problem, we analogously obtain the following result.

► **Proposition 6.** *Let  $\text{Mod}_n \subseteq K \subseteq \Gamma_n$ . The information inequality problem over  $K$  w.r.t. acyclic  $\Sigma$ -inequalities is in polynomial time.*

## 4.3 Monotone functions

Next we will show that, at the other extreme direction, the information inequality problem over monotone functions is also in polynomial time.

► **Theorem 7.** *The information inequality problem over monotone functions is in polynomial time.*

Analogously to the previous section, this semantic modification of the information inequality problem helps us identify syntactic classes with respect to which the general information inequality problem is tractable. Before we proceed into details, let us give a short sketch of the proof of this theorem. We associate an information inequality (3) with a *set representation*  $(S^+, S^-)$ , where

$$S^+ := \{(\mathbf{X}_i, k) \mid k \in [|c_i|], c_i > 0\}, \text{ and}$$

$$S^- := \{(\mathbf{X}_i, k) \mid k \in [|c_i|], c_i < 0\}.$$

Then, we present a fixed-point algorithm (Alg. 1) to capture validity of information inequalities over monotone functions. This algorithm iteratively decomposes an input inequality into monotonicity axioms. For this, it maintains a bipartite directed graph  $G$  initialized as  $G_S = (S^+ \cup S^-, E)$ , where  $E$  is the set of edges from  $S^+$  to  $S^-$  that correspond to possible monotonicity axioms (*forward edges*). The initial graph contains no edges from  $S^-$  to  $S^+$  (*backward edges*). The number of these backward edges, which represent those monotonicity axioms that are currently selected for the decomposition, is increased in each iteration. Although the algorithm as such does not run in polynomial time (it runs in *pseudo-polynomial time*, i.e., in polynomial time in the length of the input and the numeric values of the coefficients), it does guide us toward a characterization of valid inequalities as positive combinations of monotonicity axioms and *non-negativity axioms*  $h(\mathbf{X}) \geq 0$ , which are derivable as combinations of the first two polymatroidal axioms. These combinations are polynomial in the input length, and consequently can be found through a linear program of polynomial size, which entails the desired result.

■ **Algorithm 1** Decomposition algorithm for inequalities.

**Input:** Set representation  $S = (S^+, S^-)$  of  $\phi$

**Output:** true iff  $\phi$  is  $\mathcal{M}_n$ -valid

- 1:  $G \leftarrow G_S, S_0 \leftarrow S^+, S_1 \leftarrow S^-$
  - 2: **while**  $G$  contains a path  $u_0, \dots, u_m$  from  $S_0$  to  $S_1$  **do**
  - 3:     remove  $u_0$  from  $S_0$  and  $u_m$  from  $S_1$
  - 4:     remove from  $G$  (backward) edges  $(u_1, u_2), (u_3, u_4), \dots, (u_{m-2}, u_{m-1})$
  - 5:     add to  $G$  (backward) edges  $(u_1, u_0), (u_3, u_2), \dots, (u_m, u_{m-1})$
- return** true if  $S_1$  is empty, otherwise false

The following example demonstrates the use of Alg. 1.

► **Example 8.** Consider an information inequality of the form

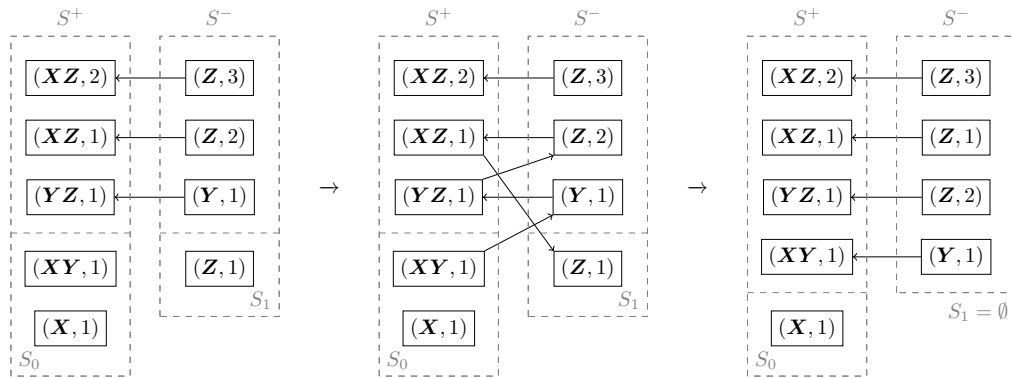
$$\mathbf{XY} + \mathbf{YZ} + 2\mathbf{XZ} + \mathbf{X} \geq \mathbf{Y} + 3\mathbf{Z}. \quad (10)$$

The set representation is  $(S^+, S^-)$ , where

$$S^+ = \{(\mathbf{XY}, 1), (\mathbf{YZ}, 1), (\mathbf{XZ}, 1), (\mathbf{XZ}, 2), (\mathbf{X}, 1)\}, \text{ and}$$

$$S^- = \{(\mathbf{Y}, 1), (\mathbf{Z}, 1), (\mathbf{Z}, 2), (\mathbf{Z}, 3)\}.$$

Clearly, the inequality (10) is valid over monotone functions. Alg. 1 also returns true after four iterations. The leftmost graph in Fig. 2 illustrates the starting point for the last iteration in one possible implementation. The edges from right to left (backward edges) represent monotonicity axioms that have been selected in the previous iteration. The edges from left to right (forward edges), some of which are visible in the middle graph of Fig. 2, represent possible monotonicity axioms. Since there is a path from  $S_0$  to  $S_1$ , we can increase the number of selected monotonicity axioms by deleting the backward edges in the path, and changing the direction of the forward edges in the path. The rightmost graph illustrates the result of this modification. Since  $S_1$  becomes empty, the algorithm terminates returning true. The final state of the algorithm represents an integral decomposition of Eq. (10) into monotonicity and non-negativity axioms.



■ **Figure 2** Last iteration of Alg. 1 for Eq. (10).

## 19:12 Information Inequality Problem over Set Functions

An inequality  $\phi$  of the form (3) can be identified with its *coefficient function*  $\mathbf{c}_\phi$ , where  $c_\phi(\mathbf{X}) = c$  if the term  $ch(\mathbf{X})$  appears in (3), and otherwise  $c_\phi(\mathbf{X}) = 0$ . We then say that  $\phi$  is a (positive) combination of inequalities  $\phi_1, \dots, \phi_n$  if  $\mathbf{c}_\phi$  is a (positive) combination of  $\mathbf{c}_{\phi_1}, \dots, \mathbf{c}_{\phi_n}$ . We furthermore say that a combination of functions  $c_1\mathbf{h}_1 + \dots + c_n\mathbf{h}_n$  is *separable* if there exist no  $i, j$  and  $\mathbf{Y}$  such that  $h_i(\mathbf{Y}) < 0$  and  $h_j(\mathbf{Y}) > 0$ , while  $c_i \neq 0 \neq c_j$ . This definition is extended to combinations of inequalities in the natural way. For example, any positive combination of  $h(A) + h(B) \geq 0$  and  $h(B) - h(AB) \geq 0$  is separable, but no positive combination of inequalities  $h(A) - h(B) \geq 0$  and  $h(A) + h(B) \geq 0$  is separable. In particular, if  $\phi$  is a positive and separable combination of monotonicity and non-negativity axioms, then  $h(\mathbf{X})$  cannot appear in the left-hand side of  $\psi_0$  and in the right-hand side of  $\psi_1$ , for any two axioms  $\psi_0$  and  $\psi_1$  appearing in the combination.

We also say that a set function  $\mathbf{h}$  is Boolean-valued if it maps every  $\mathbf{X}$  to either 0 or 1. The proof the following lemma is located in Appendix B.

► **Lemma 9.** *Let  $\phi$  be an information inequality of the form*

$$c_1h(\mathbf{X}_1) + \dots + c_kh(\mathbf{X}_k) \geq 0 \quad (c_i \in \mathbb{R}). \quad (11)$$

The following are equivalent:

1.  $\phi$  is valid over monotone functions.
2.  $\phi$  is valid over monotone, Boolean-valued functions.
3.  $\phi$  is a positive and separable combination of monotonicity and non-negativity axioms.

Since linear programming is in polynomial time, we can now establish Theorem 7 as a consequence of the following lemma. This lemma will also be applied in the next section that focuses on simple  $\Sigma$ -inequalities.

► **Lemma 10.** *For each information inequality  $\phi$  of the form*

$$c_1h(\mathbf{X}_1) + \dots + c_kh(\mathbf{X}_k) \geq d_1h(\mathbf{Y}_1) + \dots + d_lh(\mathbf{Y}_l) \quad (c_i, d_i > 0), \quad (12)$$

there exists a matrix  $M$  such that the inequality  $M\mathbf{x} \geq \mathbf{cd}$  has a solution  $\mathbf{x} \geq 0$  if and only if  $\phi$  is valid over monotone functions. In particular,  $M$  can be constructed in polynomial time from  $\phi$  (with rational coefficients).

**Proof.** Consider first a set  $\mathbf{Y}_i$  from the right-hand side of the inequality. Let  $\mathbf{X}_{i_1}, \dots, \mathbf{X}_{i_m}$  list all those sets  $\mathbf{X}_j$  from the left-hand side that contain  $\mathbf{Y}_i$  as a subset. We need to describe how the term  $d_ih(\mathbf{Y}_i)$  is distributed to monotonicity axioms. For this, define

$$x_{i_1}^i + \dots + x_{i_m}^i \geq d_i, \quad (13)$$

where  $x_j^i$  is a variable denoting the coefficient of the monotonicity axiom  $h(\mathbf{X}_j) \geq h(\mathbf{Y}_i)$ . We also need to ensure that this variable does not grow exceedingly large. Consider a set  $\mathbf{X}_j$  from the left-hand side of the inequality, and let  $\mathbf{Y}_{j_1}, \dots, \mathbf{Y}_{j_n}$  list all those sets from the right-hand side that are contained in  $\mathbf{X}_j$ .

$$x_j^{j_1} + \dots + x_j^{j_n} \leq c_j. \quad (14)$$

Combining Eqs. (13) and (14) we obtain an inequality  $M\mathbf{x} \geq \mathbf{cd}$ , where  $M$  is a  $((k+l) \times kl)$ -matrix with entries of from  $-1, 0, 1$ ,  $\mathbf{x}$  is a vector of length  $kl$ , and  $\mathbf{cd}$  (i.e.,  $\mathbf{c}$  and  $\mathbf{d}$  concatenated) is a vector of length  $k+l$ . Obviously  $M$  can be constructed in polynomial time given  $\phi$ . Moreover,  $M\mathbf{x} \geq \mathbf{cd}$  has a solution  $\mathbf{x} \geq \mathbf{0}$  if and only if  $\phi$  is a positive and separable combination of monotonicity and non-negativity axioms. The statements of the theorem then follow by Lemma 9. ◀



#### 4.4 Simple $\Sigma$ -inequalities

Let us first recall the reason why normal and general polymatroids are known to agree on the validity of simple  $\Sigma$ -inequalities. On the one hand, any such  $\Sigma$ -inequality over a variable set  $\mathbf{X}$  can be presented in the form

$$c_1 h(\mathbf{X}_1) + \cdots + c_n h(\mathbf{X}_n) \geq d_1 h(\mathbf{Y}_1) + \cdots + d_m h(\mathbf{Y}_m) \quad (c_i, d_i > 0), \quad (15)$$

where each  $\mathbf{Y}_i$  is either the full set  $\mathbf{X}$  or some singleton set  $\{X\}$ . On the other hand, every polymatroid  $\mathbf{h}$  over  $\mathbf{X}$  can be associated with a normal polymatroid  $\mathbf{f}$  over  $\mathbf{X}$  such that  $f(\mathbf{Y}) \leq h(\mathbf{Y})$  for all  $\mathbf{Y} \subseteq \mathbf{X}$ ,  $f(\mathbf{X}) = h(\mathbf{X})$ , and  $f(X) = h(X)$  for all  $X \in \mathbf{X}$  [26]. Hence, if  $\mathbf{h}$  is a counterexample for Eq. (15), then  $\mathbf{f}$  must also be a counterexample. Since normal polymatroids are positive combinations step functions, it follows that at least one step function is also a counterexample. This brings us to the following result.

► **Theorem 11** ([26]). *Let  $\phi(\mathbf{X})$  be an information inequality of the form Eq. (15), where each  $\mathbf{Y}_i$  is either the full set  $\mathbf{X}$  or a singleton set. Then,  $\phi$  is valid over step functions if and only if it is valid over polymatroids.*

Since  $S_n \subseteq N_n \subseteq \Gamma_n^* \subseteq \Gamma_n$ , it follows that validity for simple  $\Sigma$ -inequalities coincides for step functions, normal polymatroids, entropic functions, and polymatroids. If we remove terms of the form  $h(\mathbf{X})$  from the right-hand side of Eq. (15), the previous result extends to monotone functions.

► **Theorem 12.** *Let  $\phi(\mathbf{X})$  be an information inequality of the form Eq. (15), where each  $\mathbf{Y}_i$  is a singleton set. Then,  $\phi$  is valid over step functions if and only if it is valid over monotone functions.*

**Proof.** Since step functions are monotone, we only need to consider the “only-if” direction. To show the contraposition, assume that  $\phi$  is not valid over monotone functions. By Lemma 9, we find a monotone, Boolean-valued function  $h$  such that Eq. (15) becomes false. Consider the step function  $s^U$ , where  $U$  consists of all those variables  $A_i$  that are mapped to 1 by  $h$ . Clearly,  $h$  and  $s^U$  agree on the right-hand side of Eq. (15). Furthermore, for any set  $\mathbf{Z}$ , we have  $s^U(\mathbf{Z}) \leq h(\mathbf{Z})$  by monotonicity of  $h$ . Consequently, Eq. (15) is also false for  $s^U$ , meaning that  $\phi$  is not valid over step functions. ◀

It follows that validity for information inequalities of the form (15), where  $\mathbf{Y}_i$  are singletons, is decidable in polynomial time with respect to any  $K$  such that  $S_n \subseteq K \subseteq \text{Mon}_n$ , including  $K = \Gamma_n^*$ . Note that simple  $\Sigma$ -inequalities are not of this form; rewritten in the form (15) one of the sets  $\mathbf{Y}_i$  is the full variable set. However, as we will see next, it is possible to remove such terms in a single step.

Continuing our analysis of  $\phi(\mathbf{X})$  of the form (15), fix a variable  $A$  from  $\mathbf{X}$ . Define sums

$$c_A = \sum_{\substack{i \in [n] \\ A \in \mathbf{X}_i}} c_i \quad \text{and} \quad d_A = \sum_{\substack{i \in [m] \\ A \in \mathbf{Y}_i}} d_i,$$

and define the  $A$ -reduction of  $\phi$  as the inequality  $\phi^A(\mathbf{X} \setminus \{A\})$  given as

$$(c_A - d_A)h(\mathbf{X} \setminus \{A\}) + \sum_{\substack{i \in [n] \\ A \notin \mathbf{X}_i}} c_i h(\mathbf{X}_i) \geq \sum_{\substack{i \in [m] \\ A \notin \mathbf{Y}_i}} d_i h(\mathbf{Y}_i), \quad (16)$$

## 19:14 Information Inequality Problem over Set Functions

► **Lemma 13.** *An information inequality  $\phi(\mathbf{X})$  of the form (15) (having no restrictions on sets  $\mathbf{Y}_i$ ) is valid over step functions if and only if for all  $A \in \mathbf{X}$ , the  $A$ -reduction  $\phi^A$  of  $\phi$  is valid over step functions.*

1.  $c_A \geq d_A$ , and
2. the  $A$ -reduction  $\phi^A(\mathbf{X} \setminus \{A\})$  is valid over step functions.

**Proof.** Note that  $s^{\{A\}} \models \phi$  if and only if  $c_A \geq d_A$ . Also, if  $\emptyset \neq \mathbf{Y} \subseteq \mathbf{X} \setminus \{A\}$ , we have  $s^{\mathbf{Y} \cup \{A\}} \models \phi$  if and only if  $s^{\mathbf{Y}} \models \phi^A$ , where  $s^{\mathbf{Y} \cup \{A\}}$  and  $s^{\mathbf{Y}}$  refer specifically to the set functions over  $\mathbf{X}$  and  $\mathbf{X} \setminus \{A\}$ , respectively. The statement of the lemma follows. ◀

In particular, if each  $\mathbf{Y}_i$  in the inequality (15) is either a singleton or the full set  $\mathbf{X}$ , then checking validity of this inequality reduces to checking validity of a linear number of inequalities in which the sets appearing in the right-hand side are all singletons. Theorems 7, 11, and 12, and Lemma 13 thus entail that the validity of such inequalities (15) over normal polymatroids, entropic functions, and polymatroids can be determined in polynomial time. This leads us to the following corollary.

► **Corollary 14.** *The information inequality problem w.r.t. simple  $\Sigma$ -inequalities is in polynomial time over normal polymatroids, entropic functions, and polymatroids.*

One may recall from Theorem 4 that, at least in the context of step functions, the requirement of  $\Sigma$  being simple is necessary.

We conclude this section by offering an alternative proof for the fact that the entropic bound for simple sets of conditionals  $\Sigma$  is polynomial-time computable. In order to formulate this statement precisely, we need the concept of the logarithmic bound. Similarly to the degree values, the *log-degree values* associated with  $\Sigma$  are defined as a sequence  $\mathbf{b} = (b_\sigma)_{\sigma \in \Sigma}$ , where  $b_\sigma \geq 0$ . A function  $\mathbf{h}$  satisfies  $(\Sigma, \mathbf{b})$ , denoted  $\mathbf{h} \models (\Sigma, \mathbf{b})$ , if  $h(\sigma) \leq b_\sigma$  for all  $\sigma \in \Sigma$ . For a set  $S \subseteq \mathbb{R}^{2^n}$ , and a set of conditionals  $\Sigma$  guarded by a query  $Q(\mathbf{X})$  and associated with values  $\mathbf{b}$ , define the *log-bound* of  $Q$  w.r.t.  $S$  as

$$\text{Log-Bound}_S(Q, \Sigma, \mathbf{b}) := \inf_{\substack{\mathbf{w} \geq 0 \\ S \models \phi_\Sigma(\mathbf{X}, \mathbf{w})}} \sum_{\sigma \in \Sigma} w_\sigma b^\sigma,$$

where  $\phi_\Sigma(\mathbf{X}, \mathbf{w})$  is the  $\Sigma$ -inequality (9). It is known that the *entropic log-bound*  $\text{Log-Bound}_{\Gamma_n^*}$  is computable in polynomial time [11]. In the following, we present an alternative proof for this fact via monotone functions.

► **Theorem 15.** *Let  $\Sigma$  be a set of conditionals that is guarded by a query  $Q(\mathbf{X})$  and associated with values  $\mathbf{b}$ . If  $\Sigma$  is simple, the entropic log-bound  $\text{Log-Bound}_{\Gamma_n^*}(Q, \Sigma, \mathbf{b})$  is computable in polynomial time in the size of the input  $(Q, \Sigma, \mathbf{b})$ .*

**Proof.** We construct a linear program that is polynomial in the size of the input and such that its optimal value is attained at the entropic log-bound. Theorem 11 entails

$$\Gamma_n^* \models \phi_\Sigma(\mathbf{X}, \mathbf{w}) \iff S_n \models \phi_\Sigma(\mathbf{X}, \mathbf{w}), \quad (17)$$

where  $\phi_\Sigma$  is the  $\Sigma$ -inequality (9). Lemma 13 implies that

$$S_n \models \phi_\Sigma(\mathbf{X}, \mathbf{w}) \iff \forall A \in \mathbf{X} : c_A \geq d_A \text{ and } S_{n-1} \models \phi_\Sigma^A, \quad (18)$$

where  $c_A, d_A$  are the sums of coefficients  $w_\sigma$  computed from  $\phi_\Sigma$  for a variable  $A$ . Since  $\phi_\Sigma^A$  contain only singletons on their right-hand sides, Lemma 12 yields

$$S_{n-1} \models \phi_\Sigma^A \iff \text{Mon}_{n-1} \models \phi_\Sigma^A.$$

By Theorem 7 we can construct in polynomial time matrices  $M_A$  such that

$$\text{Mon}_{n-1} \models \phi_\Sigma^A \iff M_A \mathbf{x}_A \geq \mathbf{w}_A \text{ for some } \mathbf{x}_A \geq 0,$$

where  $\mathbf{w}_A$  is a list (with possible repetitions) of coefficients  $w_\sigma$  that appear in  $\phi_\Sigma^A$ . Note that we should now treat  $w_\sigma$  as variables, since we are interested in optimizing their values. Thus we rewrite  $M_A \mathbf{x}_A \geq \mathbf{w}_A \wedge c_A \geq d_A$  as  $M'_A \mathbf{x}_A \mathbf{w}_A \geq 0$ , where  $M'_A$  is obtained from  $(M_A \mid -I_{|\mathbf{w}_A|})$  by adding one extra row to describe the inequality  $c_A \geq d_A$ . Then we construct a single matrix  $M^*$  such that  $M^* \mathbf{x} \mathbf{w} = (\mathbf{x}_A \mathbf{w}_A)_{A \in \mathbf{X}}$ , where  $\mathbf{w} = (w_\sigma)_{\sigma \in \Sigma}$ , and  $\mathbf{x}$  is the concatenation of all  $\mathbf{x}_A$ . Finally, composing  $M'_A$  diagonally into a single matrix  $M_{\mathbf{X}}$ , and writing  $M_\sigma = M_{\mathbf{X}} M^*$ , we obtain

$$\forall A \in \mathbf{X} : c_A \geq d_A \text{ and } \text{S}_{n-1} \models \phi_\Sigma^A \iff M_\Sigma \mathbf{x} \mathbf{w} \geq 0. \quad (19)$$

By Eqs. (17), (18), and (19) we obtain

$$\text{Log-Bound}_{\Gamma_n^*}(Q, \Sigma, \mathbf{b}) = \inf_{\substack{\mathbf{w} \geq 0 \\ \Gamma_n^* \models \phi_\Sigma(\mathbf{X}, \mathbf{w})}} \sum_{\sigma \in \Sigma} w_\sigma b^\sigma = \min_{\substack{\mathbf{x} \mathbf{w} \geq 0 \\ M_\Sigma \mathbf{x} \mathbf{w} \geq 0}} \sum_{\sigma \in \Sigma} w_\sigma b^\sigma.$$

Since  $M_\Sigma$  can be constructed in polynomial time in the size of  $(Q, \Sigma, \mathbf{b})$ , we can compute in polynomial time the entropic log-bound  $\text{Log-Bound}_{\Gamma_n^*}(Q, \Sigma, \mathbf{b})$  as the optimal value of the linear program

$$\begin{aligned} & \text{minimize} && \sum_{\sigma \in \Sigma} w_\sigma b^\sigma \\ & \text{subject to} && M_\Sigma \mathbf{x} \mathbf{w} \geq \mathbf{0} \\ & && \mathbf{x} \mathbf{w} \geq \mathbf{0}. \end{aligned} \quad \blacktriangleleft$$

## 5 Conclusion

The present paper marks the first attempt to demarcate the tractability boundary for different variants of the information inequality problem, introduced in [14]. We established that this problem is **coNP**-complete over normal polymatroids, and in polynomial time over monotone functions. Restricted to  $\Sigma$ -inequalities where  $|\mathbf{U}| \leq 2$  for all  $(\mathbf{V} \mid \mathbf{U}) \in \Sigma$ , we proved that the information inequality problem remains **coNP**-hard over normal polymatroids. The same problem was shown to be in polynomial time over normal polymatroids, entropic functions, and polymatroids if  $|\mathbf{U}| \leq 1$ . If every set in the right-hand side of Eq. (15) is a singleton or the full variable set, we proved that the information inequality problem is in polynomial time over any  $K$  that falls inbetween normal polymatroids and monotone functions. Using this result, we constructed an alternative proof for the polynomial-time computability of the entropic bound in the case where the set of conditionals  $\Sigma$  is simple.

Based on these findings we may delineate a preliminary complexity classification of information inequalities over different set functions and syntactic classes. Consider an information inequality  $\phi$  over  $n$  variables, presented in the form (15). If  $A$  and  $B$  are subsets of  $[n]$ , we say that  $\phi$  is of *type*  $(A, B)$  if  $|\mathbf{X}_i| \in A$  for each  $\mathbf{X}_i$  appearing in the left-hand side in (15), and  $|\mathbf{Y}_j| \in B$  for each  $\mathbf{Y}_j$  appearing in the right-hand side in (15). For instance, the inequality (8) is of type  $(\{1, n\}, \{2\})$ , and all simple  $\Sigma$ -inequalities are of type  $([n], \{1, n\})$ . Using this convention, Tab. 3 summarizes the results of this paper.

Specifically, we showed that results on step functions and monotone functions lead to a polynomial-time algorithm for the entropic bound over simple degree constraints. To find more results of this kind, it may be useful to extend investigations to also other classes

■ **Table 3** Complexity of the information inequality problem for different syntactic types and set functions.

types/set functions	$\text{Mod}_n$	$\text{N}_n$	$\Gamma_n^*$	$\Gamma_n$	$\text{Mon}_n$
$([n], [n]), (\{1, n\}, \{2\})$	$\in \text{P}$ [23]	coNP-complete	$\in \Pi_1^0$ [14]	$\in \text{EXP}$ [27]	$\in \text{P}$
$([n], \{1, n\})$	$\in \text{P}$ [23]	$\in \text{P}$	$\in \text{P}$	$\in \text{P}$	$\in \text{P}$

of set functions. For instance, as illustrated in Examples 1 and 3, as  $k$  grows uniform distributions over (bags of)  $k$  tuples yield increasingly accurate answers to questions about entropic constraints, compared to step functions derived from two tuples. One way to identify more decidable classes of information inequalities would be to find syntactic restrictions for which validity is captured by uniform distributions over  $k$  tuples, for some fixed  $k$ .

---

### References

- 1 Marcelo Arenas and Leonid Libkin. An information-theoretic approach to normal forms for relational and XML data. *J. ACM*, 52(2):246–283, 2005. doi:10.1145/1059513.1059519.
- 2 Albert Atserias, Martin Grohe, and Dániel Marx. Size bounds and query plans for relational joins. *SIAM J. Comput.*, 42(4):1737–1767, 2013. doi:10.1137/110859440.
- 3 Catriel Beeri, Ronald Fagin, and John H. Howard. A complete axiomatization for functional and multivalued dependencies in database relations. In *SIGMOD Conference*, pages 47–61. ACM, 1977. doi:10.1145/509404.509414.
- 4 Randall Dougherty, Chris Freiling, and Kenneth Zeger. Non-shannon information inequalities in four random variables, 2011. doi:10.48550/arXiv.1104.3602.
- 5 E. Mark Gold. Complexity of automaton identification from given data. *Inf. Control.*, 37(3):302–320, 1978. doi:10.1016/S0019-9958(78)90562-4.
- 6 Georg Gottlob, Stephanie Tien Lee, Gregory Valiant, and Paul Valiant. Size and treewidth bounds for conjunctive queries. *J. ACM*, 59(3):16:1–16:35, 2012. doi:10.1145/2220357.2220363.
- 7 Martin Grohe and Dániel Marx. Constraint solving via fractional edge covers. *ACM Trans. Algorithms*, 11(1):4:1–4:20, 2014. doi:10.1145/2636918.
- 8 Emirhan Gürpınar and Andrei E. Romashchenko. How to use undiscovered information inequalities: Direct applications of the copy lemma. In *ISIT*, pages 1377–1381. IEEE, 2019. doi:10.1109/ISIT.2019.8849309.
- 9 Miika Hannula. Information inequality problem over set functions. *CoRR*, abs/2309.11818, 2023. doi:10.48550/arXiv.2309.11818.
- 10 Christian Herrmann. On the undecidability of implications between embedded multivalued database dependencies. *Information and Computation*, 122(2):221–235, 1995. doi:10.1006/inco.1995.1148.
- 11 Sungjin Im, Benjamin Moseley, Hung Q. Ngo, Kirk Pruhs, and Alireza Samadian. Optimizing polymatroid functions. *CoRR*, abs/2211.08381, 2022. doi:10.48550/arXiv.2211.08381.
- 12 Tarik Kaced and Andrei E. Romashchenko. Conditional information inequalities for entropic and almost entropic points. *IEEE Trans. Inf. Theory*, 59(11):7149–7167, 2013. doi:10.1109/TIT.2013.2274614.
- 13 Batya Kenig and Dan Suciú. Integrity constraints revisited: From exact to approximate implication. *Log. Methods Comput. Sci.*, 18(1), 2022. doi:10.46298/lmcs-18(1:5)2022.
- 14 Mahmoud Abo Khamis, Phokion G. Kolaitis, Hung Q. Ngo, and Dan Suciú. Decision problems in information theory. In *ICALP*, volume 168 of *LIPICs*, pages 106:1–106:20, 2020. doi:10.4230/LIPICs.ICALP.2020.106.
- 15 Mahmoud Abo Khamis, Phokion G. Kolaitis, Hung Q. Ngo, and Dan Suciú. Bag query containment and information theory. *ACM Trans. Database Syst.*, 46(3):12:1–12:39, 2021. doi:10.1145/3472391.

- 16 Mahmoud Abo Khamis, Hung Q. Ngo, and Dan Suciu. Computing join queries with functional dependencies. In *PODS*, pages 327–342. ACM, 2016. doi:10.1145/2902251.2902289.
- 17 Mahmoud Abo Khamis, Hung Q. Ngo, and Dan Suciu. What do shannon-type inequalities, submodular width, and disjunctive datalog have to do with one another? In *PODS*, pages 429–444. ACM, 2017. doi:10.1145/3034786.3056105.
- 18 Lukas Kühne and Geva Yashfe. On entropic and almost multilinear representability of matroids. *CoRR*, abs/2206.03465, 2022. doi:10.48550/arXiv.2206.03465.
- 19 Tony T. Lee. An information-theoretic analysis of relational databases - part I: data dependencies and information metric. *IEEE Trans. Software Eng.*, 13(10):1049–1061, 1987. doi:10.1109/TSE.1987.232847.
- 20 Tony T. Lee. An information-theoretic analysis of relational databases - part II: information structures of database schemas. *IEEE Trans. Software Eng.*, 13(10):1061–1072, 1987. doi:10.1109/TSE.1987.232848.
- 21 Cheuk Ting Li. Undecidability of network coding, conditional information inequalities, and conditional independence implication. *IEEE Trans. Inf. Theory*, 69(6):3493–3510, 2023. doi:10.1109/TIT.2023.3247570.
- 22 Wing Ning Li. Two-segmented channel routing is strong np-complete. *Discret. Appl. Math.*, 78(1-3):291–298, 1997. doi:10.1016/S0166-218X(97)00020-6.
- 23 Hung Q. Ngo. Worst-case optimal join algorithms: Techniques, results, and open problems. In *PODS*, pages 111–124. ACM, 2018. doi:10.1145/3196959.3196990.
- 24 Hung Q. Ngo, Ely Porat, Christopher Ré, and Atri Rudra. Worst-case optimal join algorithms. *J. ACM*, 65(3):16:1–16:40, 2018. doi:10.1145/3180143.
- 25 Nicholas Pippenger. What are the laws of information theory. In *Special Problems on Communication and Computation Conference*, pages 3–5, 1986.
- 26 Dan Suciu. Applications of information inequalities to database theory problems. In *LICS*, pages 1–30, 2023. doi:10.1109/LICS56636.2023.10175769.
- 27 Raymond W. Yeung. *Information Theory and Network Coding*. Springer Publishing Company, Incorporated, 1 edition, 2008.
- 28 Z. Zhang and R.W. Yeung. A non-shannon-type conditional inequality of information quantities. *IEEE Transactions on Information Theory*, 43(6):1982–1986, 1997. doi:10.1109/18.641561.

## A Alternative coNP-hardness proof

Recall that validity coincides for step functions and normal polymatroids, and thus it suffices to consider validity in the former sense. We reduce from three-colorability. Let  $G = (\mathbf{V}, \mathbf{E})$  be a graph consisting of a vertex set  $\mathbf{V}$  and a set of undirected edges  $\mathbf{E}$ . For each node  $A \in \mathbf{V}$ , we introduce variables  $A_r, A_g, A_b$  representing possible colors of  $A$ . Assume that the graph contains  $n$  vertices. We define

$$\sum_{\substack{c \in \{r,g,b\} \\ A \in \mathbf{V}}} h(A_c) + \sum_{\substack{c,d \in \{r,g,b\} \\ c \neq d \\ A \in \mathbf{V}}} (2n+1)h(\mathbf{V} \mid A_c A_d) + \sum_{\substack{c \in \{r,g,b\} \\ \{A,B\} \in \mathbf{E}}} (2n+1)h(\mathbf{V} \mid A_c B_c) \geq (2n+1)h(\mathbf{V}). \quad (20)$$

We claim that  $G$  is three-colorable if and only if Eq. (20) is not valid over  $S_n$ .

Assume first Eq. (20) is not valid, and let  $s_U, U \subseteq \mathbf{V}$ , be a step function such that Eq. (20) is false for  $h = s_U$ . We claim that the function that maps each vertex  $A$  to a color  $c$  if  $A_c \in U$  is well-defined and constitutes a coloring of the graph. Since the entropy and the conditional entropy are non-negative for all step functions, we have  $s_U(\mathbf{V}) = 1$ , and thus the right-hand side of Eq. (20) is  $2n + 1$ . Consequently, the left-hand side is at most  $2n$ . From the first summation term, we obtain that  $U$  must contain at least  $n$  elements. Moreover,

## 19:18 Information Inequality Problem over Set Functions

each term of the form  $h(\mathbf{V} \mid A_c A_d)$  or  $h(\mathbf{V} \mid A_c B_c)$  must be zero. In particular, we have  $A_c A_d \not\subseteq \mathcal{U}$  and  $A_c B_c \not\subseteq \mathcal{U}$ , which entails that each vertex is assigned exactly one color, and no two vertices connected by an edge are assigned the same color. We conclude that the function defined by the step function is well defined and constitutes a graph coloring.

Assume then Eq. (20) is valid. For each coloring of the vertices we may define a subset  $\mathcal{U} \subseteq \mathbf{V}$  such that  $A_c \in \mathcal{U}$  if and only if vertex  $A$  is assigned color  $c$ . Then, the first summation term in the left-hand side of Eq. (20) is  $n$ , and the second summation term is zero. By hypothesis, some term of the form  $h(\mathbf{V} \mid A_c B_c)$  must be non-zero, which means that there exists an edge whose endpoints are assigned the same color. This concludes the proof of the claim.

Since the reduction is in polynomial time, and each coefficient is bounded by a polynomial in the input size, strong coNP-completeness again follows. ◀

### B Completeness of fixed-point algorithm

► **Lemma 9.** *Let  $\phi$  be an information inequality of the form*

$$c_1 h(\mathbf{X}_1) + \cdots + c_k h(\mathbf{X}_k) \geq 0 \quad (c_i \in \mathbb{R}). \quad (11)$$

The following are equivalent:

1.  $\phi$  is valid over monotone functions.
2.  $\phi$  is valid over monotone, Boolean-valued functions.
3.  $\phi$  is a positive and separable combination of monotonicity and non-negativity axioms.

**Proof.** The implications (3)  $\Rightarrow$  (1) and (1)  $\Rightarrow$  (2) are immediate. We prove that (2)  $\Rightarrow$  (3). Clearly, if this implication holds w.r.t.  $c_i \in \mathbb{Z}$ , then it holds w.r.t.  $c_i \in \mathbb{Q}$ . We first prove the following claim.

▷ **Claim 16.** If the implication (2)  $\Rightarrow$  (3) holds w.r.t.  $c_i \in \mathbb{Q}$ , then it holds w.r.t.  $c_i \in \mathbb{R}$ .

**Proof.** To prove this, assume  $\phi$  is valid over  $\text{Mon}_n^{0,1}$ . Let  $(\phi_n)$  be a sequence of information inequalities

$$c_1^n h(\mathbf{X}_1) + \cdots + c_k^n h(\mathbf{X}_k) \geq 0 \quad (c_i^n \in \mathbb{Q}, n \geq 1), \quad (21)$$

where  $\lim_{n \rightarrow \infty} c_i^n = c_i$  and  $c_i^n \geq c_i^{n+1}$ . We may assume that  $c_i^1$  is negative if  $c_i$  is negative. That is,  $(c_i^n)$  is a sequence of positive (resp. negative) values if  $c_i$  is positive (resp. negative). Clearly, if  $\phi$  is valid over Boolean-valued, monotone functions, then so are  $\phi_n$ . By hypothesis,  $\phi_n$  decompose into positive and separable combinations of monotonicity and non-negativity axioms. Writing  $\mathbf{c}_\phi$  for the coefficient function arising from  $\phi$ , we may write

$$\mathbf{c}_{\phi_n} = d_1^n \mathbf{c}_{\psi_1} + \cdots + d_m^n \mathbf{c}_{\psi_m} \quad (d_i^n \geq 0), \quad (22)$$

where  $\psi_l$  list all possible monotonicity and non-negativity axioms respectively of the form  $h(\mathbf{X}_i) \geq 0$  and  $h(\mathbf{X}_i) - h(\mathbf{X}_j) \geq 0$ , where  $i, j \in [k]$  and  $\mathbf{X}_j \subseteq \mathbf{X}_i$ , excluding those  $\psi_l$  for which the coefficient  $d_l^n$  is always zero. That is, the combinations (22) are separable and have fixed length over all  $n \geq 1$ ; recall that separability was defined with respect to terms having a non-zero coefficient. Fix attention to an arbitrary  $\psi_l$  being either of the form  $h(\mathbf{X}_i) \geq 0$  or  $h(\mathbf{X}_i) - h(\mathbf{X}_j) \geq 0$ . In this case, the coefficient function  $\mathbf{c}_{\psi_l}$  maps  $\mathbf{X}_i$  to 1, that is,  $c_{\psi_l}(\mathbf{X}_i) = 1$ . We claim that the coefficient  $c_i$  of  $h(\mathbf{X}_i)$  in Eq. (11) is positive. For this, consider some  $p \geq 1$  such that the coefficient  $d_l^p$  of  $\mathbf{c}_{\psi_l}$  is strictly positive. Assume toward

contradiction that  $c_i$  is not positive, meaning that it is negative. Then by construction,  $c_i^p$  is negative (i.e.,  $c_{\phi_p}(\mathbf{X}_i) < 0$ ), whence  $c_{\psi_{l'}}(\mathbf{X}_i) < 0$  for some  $l' \neq l$  associated with a strictly positive coefficient  $d_{l'}^p$ . Since  $c_{\psi_l}(\mathbf{X}_i) > 0$ , this contradicts separability of (22), proving our claim. The claim entails by construction that  $c_i^n$  are positive for all  $n \geq 1$ . Hence we obtain  $d_l^n \leq c_i^n \leq c_i^1$  by separability of (22).

We conclude that  $(\mathbf{d}_n) = (d_1^n, \dots, d_m^n)$  is an infinite and bounded sequence of  $\mathbb{R}^m$ . The Bolzano-Weierstrass theorem entails that  $(\mathbf{d}_n)$  has a subsequence  $(\mathbf{d}_{n_p})$  that converges to some  $\mathbf{d} = (d_1, \dots, d_m)$ . Obviously the vector  $\mathbf{d}$  is non-negative. By continuity,

$$\mathbf{c}_\phi = \lim_{p \rightarrow \infty} \mathbf{c}_{\phi_{n_p}} = \lim_{p \rightarrow \infty} d_1^{n_p} \mathbf{c}_{\psi_1} + \dots + d_m^{n_p} \mathbf{c}_{\psi_m} = d_1 \mathbf{c}_{\psi_1} + \dots + d_m \mathbf{c}_{\psi_m}.$$

The obtained combination is separable, because otherwise some combination (22) is not separable for large enough  $n_p$ , which leads to a contradiction. We conclude that  $\phi$  is a positive and separable combination of monotonicity and non-negativity axioms, which shows that (2)  $\Rightarrow$  (3) w.r.t.  $c_i \in \mathbb{R}$ .  $\triangleleft$

It remains to prove that (2)  $\Rightarrow$  (3) w.r.t.  $c_i \in \mathbb{Z}$ . Let  $S = (S^+, S^-)$  be the set representation of  $\phi$ . We associate  $S$  with a directed graph  $G_S$ , where

- the set of nodes are the elements of  $S^+$  and  $S^-$ , and
- there is a directed edge from  $(\mathbf{X}, i)$  to  $(\mathbf{Y}, j)$  if  $(\mathbf{X}, i) \in S^+$ ,  $(\mathbf{Y}, j) \in S^-$ , and  $\mathbf{Y} \subseteq \mathbf{X}$ .

Consider Alg. 1 which maintains a bipartite directed graph  $G$  that is initially set up as  $G_S$ . The monotonicity axioms isolated at the current step are represented as directed edges going from  $S^-$  to  $S^+$  *backward edges*; in the beginning no such edges have been introduced yet. The edges that proceed from  $S^+$  to  $S^-$  (*forward edges*) are kept fixed.

We say that a node  $u$  is *connected* to a node  $v$  in a directed graph if  $u = v$ , or there is a sequence of nodes (a *path* from  $u$  to  $v$ ) in which the first node is  $u$ , the last node is  $v$ , and each node is connected to the following node by a directed edge. A set  $\mathbf{U}$  is *connected* to another set  $\mathbf{V}$  if some node in  $\mathbf{U}$  is connected to some node in  $\mathbf{V}$ .

Consider the following claim.

$\triangleright$  **Claim 17.** If Alg. 1 returns true on  $\phi$ , then  $\phi$  is a positive and separable combination of the monotonicity and non-negativity axioms.

*Proof.* Consider the graph  $G$  and the sets  $S_0$  and  $S_1$  after termination of the algorithm. Note that if  $G$  contains a backward edge  $(u, v)$ , then the reverse edge  $(v, u)$  forms a forward edge of  $G_S$  and consequently corresponds to a monotonicity axiom. The backward edges also form a bijection from  $S^- \setminus S_1$  to  $S^+ \setminus S_0$ . Since  $S_1$  is empty by assumption, and  $S_0$  can be viewed as representing non-negativity axioms, it can now be observed that  $\phi$  decomposes into a positive and separable combination of monotonicity and non-negativity axioms. This proves the claim.  $\triangleleft$

We now prove the contraposition of (2)  $\Rightarrow$  (3) w.r.t.  $c_i \in \mathbb{Z}$ . Suppose  $\phi$  is not a positive and separable combination of the monotonicity axioms. The previous claim entails that the algorithm returns false. Consider again the graph  $G$  and the sets  $S_0, S_1$  after termination of the algorithm. Note that  $S_1$  is now non-empty. Let  $\mathbf{V}$  denote the set of variables appearing in  $\phi$ . Let  $\mathcal{Y}$  be the (non-empty) collection of sets  $\mathbf{Y} \subseteq \mathbf{V}$  such that for some  $j$ ,  $(\mathbf{Y}, j)$  belongs to  $S^-$  and is connected to  $S_1$ . Consider also its *upper closure*  $\mathcal{Y}^\uparrow := \{\mathbf{Z} \subseteq \mathbf{V} \mid \exists \mathbf{Y} \in \mathcal{Y} : \mathbf{Y} \subseteq \mathbf{Z}\}$ . Define a mapping  $h$  such that  $h(\mathbf{Z}) = 1$  if  $\mathbf{Z} \in \mathcal{Y}^\uparrow$ , and otherwise  $h(\mathbf{Z}) = 0$ . Clearly,  $h$  is a Boolean, monotone function. We show that  $h$  does not satisfy  $\phi$ .



## 19:20 Information Inequality Problem over Set Functions

Consider a pair  $(\mathbf{X}, i) \in S^+$  such that  $h(\mathbf{X}) = 1$ . Then,  $\mathbf{X}$  contains a set  $\mathbf{Y}$  from  $\mathcal{Y}$ . Let  $j$  be such that  $(\mathbf{Y}, j)$  belongs to  $S^-$  and is connected to  $S_1$ . Since there is an edge from  $(\mathbf{X}, i)$  to  $(\mathbf{Y}, j)$ , it follows that  $(\mathbf{X}, i)$  is connected to  $S_1$ . Now, if  $(\mathbf{X}, i)$  belonged to  $S_0$ , the algorithm could not have terminated yet. Hence  $(\mathbf{X}, i)$  must belong to  $S^+ \setminus S_0$ . Recall that the backward edges form a bijection from  $S^- \setminus S_1$  to  $S^+ \setminus S_0$ . In particular,  $(\mathbf{X}, i)$  is the target of a unique backward edge with a source node  $(\mathbf{Z}, k)$ . Since  $(\mathbf{X}, i)$  is connected to  $S_1$ , it follows that  $(\mathbf{Z}, k)$  is also connected to  $S_1$ . This entails that  $h(\mathbf{Z}) = 1$ . In particular, this shows that any  $(\mathbf{X}, i) \in S^+$  such that  $h(\mathbf{X}) = 1$  is paired by a backward edge with a unique  $(\mathbf{Z}, k) \in S^- \setminus S_1$  such that  $h(\mathbf{Z}) = 1$ . In addition, because  $S_1$  is non-empty, there exists an element  $(\mathbf{U}, l) \in S^- \cap S_1$  such that  $h(\mathbf{U}) = 1$ . In particular,  $(\mathbf{U}, l)$  is not the source node of any backward edge. These observations entail that  $h$  does not satisfy  $\phi$ . This proves the contraposition of (2)  $\Rightarrow$  (3) w.r.t.  $c_i \in \mathbb{Z}$ .

This concludes the proof of the direction (2)  $\Rightarrow$  (3).  $\blacktriangleleft$

The following example demonstrates that Alg. 1 correctly returns false on the submodularity axiom, as this axiom is not a consequence of monotonicity and non-negativity.

**► Example 18.** The submodularity axiom  $XY + XZ - X - XYZ \geq 0$  is not valid over monotone functions. This can be also seen by referring to Alg. 1. The set representation is  $(S^+, S^-)$  where  $S^+ = \{(XY, 1), (XZ, 1)\}$  and  $S^- = \{(X, 1), (XYZ, 1)\}$ . Suppose at the first step the algorithm introduces a backward edge from  $(X, 1)$  to  $(XY, 1)$ ; the only other option is the symmetric scenario where it introduces an edge from  $(X, 1)$  to  $(XZ, 1)$ . After the first step we have  $S_0 = \{(XZ, 1)\}$  and  $S_1 = \{(XYZ, 1)\}$ . Then, no path exists from  $S_0$  to  $S_1$ , since no forward edge points to  $(XYZ, 1)$ . The algorithm therefore terminates returning false. Accordingly, the function that maps  $XYZ$  to 1 and all other sets to 0 is monotone, Boolean-valued, and does not satisfy the aforementioned submodularity axiom.

# Conditional Independence on Semiring Relations

Miika Hannula  

University of Helsinki, Finland

---

## Abstract

Conditional independence plays a foundational role in database theory, probability theory, information theory, and graphical models. In databases, a notion similar to conditional independence, known as the (embedded) multivalued dependency, appears in database normalization. Many properties of conditional independence are shared across various domains, and to some extent these commonalities can be studied through a measure-theoretic approach. The present paper proposes an alternative approach via semiring relations, defined by extending database relations with tuple annotations from some commutative semiring. Integrating various interpretations of conditional independence in this context, we investigate how the choice of the underlying semiring impacts the corresponding axiomatic and decomposition properties. We specifically identify positivity and multiplicative cancellativity as the key semiring properties that enable extending results from the relational context to the broader semiring framework. Additionally, we explore the relationships between different conditional independence notions through model theory.

**2012 ACM Subject Classification** Theory of computation → Database theory

**Keywords and phrases** semiring, conditional independence, functional dependency, decomposition, axiom

**Digital Object Identifier** 10.4230/LIPIcs.ICDT.2024.20

**Related Version** *Full Version*: <https://arxiv.org/abs/2310.01910> [13]

**Funding** *Miika Hannula*: The author has been supported by the ERC grant 101020762.

## 1 Introduction

Conditional independence (CI) is an expression of the form  $Y \perp\!\!\!\perp Z \mid X$ , stating that  $Y$  and  $Z$  are conditionally independent given  $X$ . Common to its different interpretations is that conditional independence is a mark of redundancy. For instance, on a relation schema over attributes  $X, Y, Z$ , the multivalued dependency (MVD)  $X \twoheadrightarrow Y$  can be viewed as the counterpart of the CI  $Y \perp\!\!\!\perp Z \mid X$ , expressing that a relation can be losslessly decomposed into its projections on  $X, Y$  and  $X, Z$ . The process of splitting the schema into smaller parts – in order to avoid data redundancy – is called normalization, and a database schema is in fourth normal form if every non-trivial MVD follows from some key. In probability theory, CIs over random variables give rise to factorizations of joint probability distributions into conditional distributions. Since the decomposed distributions can be represented more compactly, this allows more efficient reasoning about the random variables. In addition to these classical examples, conditional independence has applications in ordinal conditional functions [24], Dempster-Schaefer theory [8, 23], and possibility theory [29].

Since the notion of conditional independence has a relatively fixed meaning across various contexts, it is no coincidence that the central rules governing its behavior are universally shared. The semigraphoid axioms [21] state five basic rules that hold true for diverse interpretations of conditional independence. Initially conjectured to be complete by Pearl, Studený [25] proved incompleteness of these rules by discovering a new rule that is not derivable by the semigraphoid axioms, while being sound for probability distributions. Later he [26] proved that there cannot be any finite axiomatization for conditional independence, a fact that had been established earlier for embedded multivalued dependencies (EMVDs) [15].



© Miika Hannula;

licensed under Creative Commons License CC-BY 4.0

27th International Conference on Database Theory (ICDT 2024).

Editors: Graham Cormode and Michael Shekelyan; Article No. 20; pp. 20:1–20:20

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The *implication problem*, which is to determine whether some set of dependencies  $\Sigma$  logically implies a dependency  $\tau$ , is in fact undecidable not only for EMVDs [14], but also for CIs in probability theory, as has been recently shown [18, 20].

In some partial cases, the semigraphoid axioms are known to be complete. A *saturated conditional independence* (SCI) is a CI that contains all the variables of the underlying joint distribution. The semigraphoid axioms are complete for the implication of arbitrary CIs by saturated ones under various semantics [12], and for the implication of CIs from a set of CIs encoded in the topology of a Bayesian network [10]. In databases, where SCIs correspond to MVDs, the implication problem for MVDs combined with functional dependencies (FDs) is well-known to have a finite axiomatization and a polynomial-time algorithm [4].

Moving beyond saturated CIs, the implication problem not only becomes undecidable, but also more sensitive to the underlying semantics. Studený [27] presents several example inference rules that involve non-saturated CIs and are sound in one setting while failing to be sound in others. For instance, the aforementioned rule<sup>1</sup> showing incompleteness of the semigraphoid axioms is not sound for database relations, but its soundness for probability distributions follows by a simple information-theoretic argument. FDs and MVDs can also be alternatively expressed in terms of information measures over a uniformly distributed database relation [19], and their implication problem has recently been connected to validity of information inequalities [16]. Galliani and Väänänen [9] associate relations with a so-called diversity measure to capture FDs and other data dependencies. These measure-theoretic approaches, however, fail to capture the semantics of the embedded multivalued dependency in full generality.

This paper examines  $K$ -relations as a unifying framework for conditional independence and other dependency concepts. Introduced in the seminal work [11],  $K$ -relations extend ordinary relations by tuple annotations from a commutative semiring  $K$ , providing a powerful abstraction for data provenance. While it is natural to consider propagation of tuple annotations through queries in this context, one can also ask how tuple annotations couple with data dependencies. Dependencies on  $K$ -relations have thus far received limited attention (see, e.g., [2, 3, 7]). Related to this work, Barlag et al. [3] define conditional independence for  $K$ -relations, and raise the question of how much the related axiomatic properties depend on the algebraic properties of  $K$ . Atserias and Kolaitis [2] study the relationship between local and global consistency for  $K$ -relations, introducing also many concepts that will be adopted in this paper. Although the authors do not consider conditional independence, they show that functional dependencies on  $K$ -relations entail lossless-join decompositions.

The following contributions are presented in this paper: First, we show that conditional independence for  $K$ -relations corresponds to lossless-join decompositions whenever  $K$  is positive and multiplicatively cancellative. Then, we provide a proof that, for any  $K$  exhibiting these characteristics, the semigraphoid axioms are sound for general CIs, and extend to a complete axiomatization of SCI+FD which is comparable to that of MVD+FD. This entails that database normalization techniques extend to  $K$ -relations whenever positivity and multiplicative cancellativity are assumed. To showcase potential applications, we illustrate through an example how the semiring perspective can lead to decompositions of data tables which appear non-decomposable when interpreted relationally. Lastly, we explore how  $K$ -relations and model theory can shed light into the interconnections among different CI semantics.

---

<sup>1</sup> This rule states that  $A \perp\!\!\!\perp B \mid CD \wedge C \perp\!\!\!\perp D \mid A \wedge C \perp\!\!\!\perp D \mid B \wedge A \perp\!\!\!\perp B \mid \emptyset$  if and only if  $C \perp\!\!\!\perp D \mid AB \wedge A \perp\!\!\!\perp B \mid C \wedge A \perp\!\!\!\perp B \mid D \wedge C \perp\!\!\!\perp D \mid \emptyset$ . For probability distributions the rule follows by the non-negativity of conditional mutual information  $I(Y; Z|C)$ , and the fact that  $I(Y; Z|X) = 0$  if and only if  $Y$  and  $Z$  are conditionally independent given  $X$ . For database relations the rule is not sound; see a counterexample in [27].

## 2 Semirings

We commence by recapitulating concepts related to semirings. A *semiring* is a tuple  $K = (K, \oplus, \otimes, 0, 1)$ , where  $\oplus$  and  $\otimes$  are binary operations on  $K$ ,  $(K, \oplus, 0)$  is a commutative monoid with identity element 0,  $(K, \otimes, 1)$  is a monoid with identity element 1,  $\otimes$  left and right distributes over  $\oplus$ , and  $x \otimes 0 = 0 = 0 \otimes x$  for all  $x \in K$ . The semiring  $K$  is called *commutative* if  $(K, \otimes, 1)$  is a commutative monoid. That is, semirings are rings which need not have additive inverses. As usual, we often write  $ab$  instead of  $a \otimes b$ . In this paper, we assume that every semiring is non-trivial ( $0 \neq 1$ ) and commutative. The symbols  $\oplus, \otimes, \oplus, \otimes$  are used in reference to specific semiring operations, and symbols  $+, \cdot, \sum, \prod$  refer to ordinary arithmetic operations.

We list some example semirings that will be considered in this paper.

- The *Boolean semiring*  $\mathbb{B} = (\mathbb{B}, \vee, \wedge, 0, 1)$  models logical truth and is formed from the two-element Boolean algebra. It is the simplest example of a semiring that is not a ring.
- The *probability semiring*  $\mathbb{R}_{\geq 0} = (\mathbb{R}_{\geq 0}, +, \cdot, 0, 1)$  consists of the non-negative reals with standard addition and multiplication.
- The *semiring of natural numbers*  $\mathbb{N} = (\mathbb{N}, +, \cdot, 0, 1)$  consists of natural numbers with their usual operations.
- The *tropical semiring*  $\mathbb{T} = (\mathbb{R} \cup \{\infty\}, \min, +, \infty, 0)$  consists of the reals expanded with infinity and has min and standard addition respectively plugged in for addition and multiplication.
- The *Viterbi semiring*  $\mathbb{V} = ([0, 1], \max, \cdot, 0, 1)$  associates the unit interval with maximum as addition and standard multiplication.

Other examples include the semiring of multivariate polynomials  $\mathbb{N}[\mathbf{X}] = (\mathbb{N}[\mathbf{X}], +, \cdot, 0, 1)$  which is the free commutative semirings generated by the indeterminates in  $\mathbf{X}$ , and the *Lukasiewicz semiring*  $\mathbb{L} = ([0, 1], \max, \cdot, 0, 1)$ , used in multivalued logic, which endows the unit interval with max addition and multiplication  $a \cdot b := \max(0, a + b - 1)$ .

Let  $\leq$  be a partial order. A binary operator  $*$  is said to be *monotone under  $\leq$*  if  $a \leq b$  and  $a' \leq b'$  implies  $a * a' \leq b * b'$ . If  $*$  =  $\oplus$  (resp.  $*$  =  $\otimes$ ), we call this property of  $(K, \leq)$  *additive monotony* (resp. *multiplicative monotony*). A *partially ordered semiring* is a tuple  $K = (K, \oplus, \otimes, 0, 1, \leq)$ , where  $(K, \oplus, \otimes, 0, 1)$  is a semiring, and  $(K, \leq)$  is a partially ordered set satisfying additive and multiplicative monotony. Given a semiring  $K = (K, \oplus, \otimes, 0, 1)$ , define a binary relation  $\leq_K$  on  $K$  as

$$a \leq_K b := \Leftrightarrow \exists c : a \oplus c = b. \quad (1)$$

This relation is a preorder, meaning it is reflexive and transitive. If  $\leq_K$  is also antisymmetric, it is a partial order, called the *natural order* of  $K$ , and  $K$  is said to be *naturally ordered*. In this case,  $K$  endowed with its natural order is a partially ordered semiring. If additionally the natural order is *total*, i.e.,  $a \leq_K b$  or  $b \leq_K a$  for all  $a, b \in K$ , we say that  $K$  is *naturally totally ordered*.

If a semiring  $K$  satisfies  $ab = 0$  for some  $a, b \in K$  where  $a \neq 0 \neq b$ , we say that  $K$  has *divisors of 0*. The semiring  $K$  is called  $\oplus$ -*positive* if  $a \oplus b = 0$  implies that  $a = b = 0$ . If  $K$  is both  $\oplus$ -positive and has no divisors of 0, it is called *positive*. For example, the modulo two integer semiring  $\mathbb{Z}_2$  is not positive since it is not  $\oplus$ -positive (even though it has no divisors of 0). Conversely, an example of a semiring with divisors of 0 is  $\mathbb{Z}_4$ . A semiring is called *additively* (resp. *multiplicatively*) *cancellative* if  $a \oplus b = a \oplus c$  implies  $b = c$  (resp.  $ab = ac$  and  $a \neq 0$  implies  $b = c$ ). It is simply *cancellative* if it is both additively and multiplicatively cancellative. A semiring  $K$  in which each non-zero element has a multiplicative inverse is called a *semifield*. A semifield  $K$  in which each element has an additive inverse is a *field*.

## 20:4 Conditional Independence on Semiring Relations

In particular, note that the probability semiring  $\mathbb{R}_{\geq 0}$ , the semiring of natural numbers  $\mathbb{N}$ , the Boolean semiring  $\mathbb{B}$ , and the tropical semiring are positive, multiplicatively cancellative, and naturally ordered. Of these only the first two are also additively cancellative. This difference seems to be crucial for the behavior of conditional independence.

This section concludes with two lemmata. The first lemma is applied when examining the relationship between lossless-join decompositions and conditional independence (Theorem 10). The second lemma comes into play when comparing the CI implication problem for different semirings (Theorem 20). A formal definition of an *embedding* of a model into another model is located in Appendix A. The lemma proofs can be found in the arXiv version [13]. A field  $F$  endowed with a total order  $\leq$  is a *totally ordered field* if  $(F, \leq)$  satisfies additive monotony and *monotony of non-negative multiplication*:  $a \geq 0$  and  $b \geq 0$  implies  $ab \geq 0$ .

► **Lemma 1.** *Any positive multiplicatively cancellative semiring  $K$  embeds in a positive semifield  $F$ . Furthermore, if  $K$  is additively cancellative, then  $F$  is additively cancellative, and if  $K$  is a naturally totally ordered, then  $F$  is naturally totally ordered.*

► **Lemma 2.** *Any naturally totally ordered cancellative semiring embeds in a totally ordered field.*

### 3 $K$ -relations

This section introduces ordinary relations as well as  $K$ -relations and their associated basic properties.

We use boldface letters to denote sets. For two sets  $\mathbf{X}$  and  $\mathbf{Y}$ , we write  $\mathbf{XY}$  to denote their union. If  $A$  is an individual element, we sometimes write  $A$  instead of  $\{A\}$  to denote the singleton set consisting of  $A$ .

#### 3.1 Relations

Fix disjoint countably infinite sets  $\mathbf{Var}$  and  $\mathbf{Val}$  of variables and values. Each variable  $A \in \mathbf{Var}$  is associated with a subset of  $\mathbf{Val}$ , called the *domain* of  $A$  and denoted  $\text{Dom}(A)$ . Given a finite set of variables  $\mathbf{X}$ , an  $\mathbf{X}$ -tuple is a mapping  $t : \mathbf{X} \rightarrow \mathbf{Val}$  such that  $t(A) \in \text{Dom}(A)$ . We write  $\text{Tup}(\mathbf{X})$  for the set of all  $\mathbf{X}$ -tuples. Note that  $\text{Tup}(\emptyset)$  is a singleton set consisting of the empty tuple. For  $\mathbf{Y} \subseteq \mathbf{X}$ , the *projection*  $t[\mathbf{Y}]$  of  $t$  on  $\mathbf{Y}$  is the unique  $\mathbf{Y}$ -tuple that agrees with  $t$  on  $\mathbf{X}$ . In particular,  $t[\emptyset]$  is always the empty tuple.

A *relation*  $R$  over  $\mathbf{X}$  is a subset of  $\text{Tup}(\mathbf{X})$ . The variable set  $\mathbf{X}$  is also called the (*relation*) *schema* of  $R$ . We sometimes write  $R(\mathbf{X})$  instead of  $R$  to emphasize that  $\mathbf{X}$  is the schema of  $R$ . For  $\mathbf{Y} \subseteq \mathbf{X}$ , the *projection* of  $R$  on  $\mathbf{Y}$ , written  $R[\mathbf{Y}]$ , is the set of all projections  $t[\mathbf{Y}]$  where  $t \in R$ . A *database*  $D$  is a finite collection of relations  $\{R_1[\mathbf{X}_1], \dots, R_n[\mathbf{X}_n]\}$ . Unless stated otherwise, we assume that each relation is finite.

#### 3.2 $K$ -relations

Fix a semiring  $K$ , and let  $\mathbf{X}$  be a set of variables. A  $K$ -relation over  $\mathbf{X}$  is a function  $R : \text{Tup}(\mathbf{X}) \rightarrow K$ . Again, the variable set  $\mathbf{X}$  is called the (*relation*) *schema* of  $R$ , and we can write  $R(\mathbf{X})$  instead of  $R$  to emphasize that  $\mathbf{X}$  is the schema of  $R$ . If  $K$  is the Boolean semiring  $\mathbb{B}$ , the tuple annotation  $R(t)$  characterizes an ordinary relation, and thus we will often in this paper identify  $\mathbb{B}$ -relations and relations. Note that a  $K$ -relation over  $\emptyset$  associates the empty tuple with some value of  $K$ . The *support*  $\text{Supp}(R)$  of a  $K$ -relation  $R$  over  $\mathbf{X}$  is the set  $\{t \in \text{Tup}(\mathbf{X}) \mid R(t) \neq 0\}$  of tuples associated with a non-zero value. We often write

$R'$  for the support of  $R$ . The  $K$ -relation  $R$  is called *total* if for all  $t \in \text{Tup}(\mathbf{X})$  it holds that  $R(t) \neq 0$ , i.e., if  $\text{Supp}(R) = \text{Tup}(\mathbf{X})$ . It is called *normal* if  $\bigoplus_{t \in \text{Tup}(\mathbf{X})} R(t) = 1$ . For  $a \in K$ , we write  $aR$  for the  $K$ -relation over  $\mathbf{X}$  defined by  $(aR)(t) = aR(t)$ . For a  $\mathbf{Y}$ -tuple  $t$ , where  $\mathbf{Y} \subseteq \mathbf{X}$ , the *marginal* of  $R$  over  $t$  is defined as

$$R(t) := \bigoplus_{\substack{t' \in \text{Tup}(\mathbf{X}) \\ t'[\mathbf{Y}] = t}} R(t'). \quad (2)$$

We then write  $R[\mathbf{Y}]$  for the relation over  $\mathbf{Y}$ , called the *marginal* of  $R$  on  $\mathbf{Y}$ , that consists of the marginals of  $R$  over all  $\mathbf{Y}$ -tuples. Note that the marginal  $R[\emptyset]$  of  $R$  on the empty set is a function that maps the empty tuple to  $\sum_{t \in \text{Tup}(\mathbf{X})} R(t)$ . In particular, if  $K$  is the Boolean semiring  $\mathbb{B}$ , the marginal of  $R$  on  $\mathbf{Y}$  is the projection of  $R$  on  $\mathbf{Y}$ . In this paper, we assume that each relation is finite and non-empty, and likewise each  $K$ -relation is assumed to have a finite and non-empty support.

$K$ -relations instantiated in different ways lead to familiar notions. For instance, a database relation can be viewed as  $\mathbb{B}$ -relation, and a probability distribution as a normal  $\mathbb{R}_{\geq 0}$ -relation. Alternatively, database relations can be transformed to  $K$ -relations by reinterpreting variables as tuple annotations.

► **Example 3.** Tab. 1 collects data about room prizes in a hotel. The table can be viewed as a standard database relation. Since Price is a function of Room, Date, and Persons, one can also interpret it as a  $K$ -relation  $\text{Price}(\text{Room}, \text{Date}, \text{Persons})$  over some semiring  $K$  containing positive integers. In principle, other variables such as Room and Persons can also be turned into annotations.

■ **Table 1** Price data for hotel rooms.

Room	Date	Persons	Price
double	2023-12-01	1	100
double	2023-12-01	2	120
double	2023-08-20	1	120
double	2023-08-20	2	140
twin	2023-08-20	1	110
twin	2023-08-20	2	120

### 3.3 Basic properties

Prior to delving into the concept of conditional independence, we here list some basic properties regarding projections and supports of  $K$ -relations. Lemmata 4 and 5 appear in [2], with the exception that there  $K$  is always assumed to be positive. Also the concept of a marginal in that paper is stated otherwise as in Eq. (2), except that there  $t'$  ranges over  $R'$  instead of  $\text{Tup}(\mathbf{X})$ . Obviously the two versions lead to the same concept. To account for these slight modifications, we include the proofs of these two lemmata in the arXiv version [13].

► **Lemma 4.** *Let  $R(\mathbf{X})$  be a  $K$ -relation, and let  $\mathbf{Z} \subseteq \mathbf{Y} \subseteq \mathbf{X}$ . The following statements hold:*

1. *Assuming  $K$  is  $\oplus$ -positive, for all  $\mathbf{Y} \subseteq \mathbf{X}$  it holds that  $R'[\mathbf{Y}] = R[\mathbf{Y}]'$ .*
2. *For all  $\mathbf{Z} \subseteq \mathbf{Y} \subseteq \mathbf{X}$  it holds that  $R[\mathbf{Y}][\mathbf{Z}] = R[\mathbf{Z}]$ .*

Two  $K$ -relations  $R$  and  $R'$  over a variable set  $V$  are said to be *equivalent (up to normalization)*, written  $R \equiv R'$ , if there are  $a, b \in K \setminus \{0\}$  such that  $aR = bR'$ .

► **Lemma 5.** *Let  $K$  be a semiring, let  $W, V, W \subseteq V$ , be two variable sets, and let  $R, R', R''$  be three  $K$ -relations over  $V$ . Then,*

1.  $R \equiv R'$  implies  $R[W] \equiv R'[W]$ ; and
2. if  $K$  has no divisors of zero,  $R \equiv R'$  and  $R' \equiv R''$  implies  $R \equiv R''$ .

#### 4 Conditional independence and decompositions

Regardless of the context, what we call conditional independence tends to describe essentially the same property. For a “system” consisting of three components  $X, Y, Z$ , we might say that  $Y$  is conditionally independent of  $Z$  given  $X$  if  $Y$  does not reveal anything about  $Z$ , once  $X$  has been fixed. This usually entails that the “system” can be decomposed to its “subsystems” over  $X, Y$  and  $X, Z$  without loss of information. In this section we consider a general semantics for conditional independence over  $K$ -relations, and show that under certain assumptions, this definition matches the above intuition.

► **Definition 6** (Conditional independence for  $K$ -relations [3]). *Let  $R$  be a  $K$ -relation over a variable set  $V$ , and let  $X, Y, Z$  be disjoint subsets of  $V$ . An expression of the form  $Y \perp\!\!\!\perp Z \mid X$  is called a conditional independence (CI). We say that  $R$  satisfies  $Y \perp\!\!\!\perp Z \mid X$ , denoted  $R \models Y \perp\!\!\!\perp Z \mid X$ , if for all  $V$ -tuples  $t$ ,*

$$R(t[XY])R(t[XZ]) = R(t[XYZ])R(t[X]). \quad (3)$$

Fix a relation schema  $V$  and three pairwise disjoint subsets  $X, Y, Z \subseteq V$ . A *saturated conditional independence* (SCI) is a CI of the form  $Y \perp\!\!\!\perp Z \mid X$ , where  $XYZ = V$ . Over  $\mathbb{B}$ -relations SCIs coincide with *multivalued dependencies* (MVDs), which are expressions of the form  $X \twoheadrightarrow Y$ , where  $X$  and  $Y$  may overlap. A  $V$ -relation  $R$  *satisfies*  $X \twoheadrightarrow Y$ , written  $R \models X \twoheadrightarrow Y$ , if for all two tuples  $t, t' \in R$  such that  $t[X] = t'[X]$  there exists a third tuple  $t'' \in R$  such that  $t''[XY] = t'[XY]$  and  $t''[V \setminus XY] = t[V \setminus XY]$ . An *embedded multivalued dependency* (EMVD) is an expression of the form  $X \twoheadrightarrow Y \mid Z$ , where  $X, Y, Z$  may overlap. We say that  $R$  *satisfies*  $X \twoheadrightarrow Y \mid Z$ , written  $R \models X \twoheadrightarrow Y \mid Z$ , if the projection  $R[XYZ]$  satisfies the MVD  $X \twoheadrightarrow Y$ .

► **Example 7.** Returning to Example 3, we observe that the price function  $\text{Price}(\text{Room}, \text{Date}, \text{Persons})$  exhibits certain types of dependencies between its arguments. The room prices vary depending on the date and the room type. Additionally, adding a second person incurs a price increase by a flat rate which is independent of the date but depends on the room type. This kind of independence can be captured by viewing the price function as a  $\mathbb{T}$ -relation, in which case it satisfies the SCI  $\text{Date} \perp\!\!\!\perp \text{Persons} \mid \text{Room}$ . Suppose instead of a flat price increase, the addition of a second person incurs a 20% price increase for double rooms, and a 10% price increase for twin rooms. Then, interpreting  $\text{Price}(\text{Room}, \text{Date}, \text{Persons})$  as a  $\mathbb{R}_{\geq 0}$ -relation, we again obtain  $\text{Price} \models \text{Date} \perp\!\!\!\perp \text{Persons} \mid \text{Room}$ . When Tab. 1 is viewed as an ordinary relation, it satisfies the EMVD  $\text{Room} \twoheadrightarrow \text{Date} \mid \text{Persons}$ , while failing to satisfy any MVD.

Several conditional independence notions from the literature can be recovered through  $K$ -relations. For instance, beside EMVDs, the following examples were considered in [27] and can now be restated using the previous definition.



- For  $K = \mathbb{R}_{\geq 0}$ , the definition coincides with the concept of conditional independence in probability theory.
- For  $K = \mathbb{T}$ , the definition corresponds to conditional independence over natural conditional functions. A *natural conditional function* is a mapping  $f: \text{Tup}(\mathbf{X}) \rightarrow \mathbb{N}$ , where  $\min_{t \in \text{Tup}(\mathbf{X})} f(t) = 0$ . The notion of conditional independence over such functions [27] coincides with Def. 6 over integral-valued, total, and normal  $\mathbb{T}$ -relations. Recall that for (min-plus) tropical semirings, addition is interpreted as minimum, and multiplication as the usual addition, meaning that its neutral element is 0.
- For  $K = \mathbb{V}$ , the definition corresponds to conditional independence over possibility functions. A *possibility function* is a function  $f: \text{Tup}(\mathbf{X}) \rightarrow [0, 1]$ , where  $\sum_{t \in \text{Tup}(\mathbf{X})} f(t) = 1$ . Such functions can be viewed as normal  $\mathbb{V}$ -relations, where  $\mathbb{V}$  is the Viterbi semiring, in which case their notion of conditional independence [27] matches Def. 6.

In order to connect conditional independence over  $K$ -relations to decompositions, we next consider the concept of a join. An arguably reasonable expectation is that whenever a  $K$ -relation  $T(ABC)$  satisfies a CI  $A \perp\!\!\!\perp C \mid B$ , then one should be able to retrieve  $T$  from its projections on  $AB$  and  $BC$  using the join. That is,  $T$  should be equivalent to the join of  $T[AB]$  and  $T[BC]$  up to normalization. In the relational context this is indeed the outcome once  $A \perp\!\!\!\perp C \mid B$  is interpreted as the MVD  $B \twoheadrightarrow A$ , and the join  $R \bowtie S$  of two relations  $R(\mathbf{X})$  and  $S(\mathbf{Y})$  is given in the usual way, i.e., as the relation consisting of those  $\mathbf{XY}$ -tuples  $t$  whose projections  $t[\mathbf{X}]$  and  $t[\mathbf{Y}]$  appear respectively in  $R$  and  $S$ . In the context of  $K$ -relations the join of  $R(\mathbf{X})$  and  $S(\mathbf{Y})$  is often defined via multiplication as the  $K$ -relation  $R * S$  over  $\mathbf{XY}$  where

$$(R * S)(t) = R(t[\mathbf{X}])S(t[\mathbf{Y}]) \quad (4)$$

(see, e.g., [11]). Substituting  $K = \mathbb{B}$  in this definition now yields the standard relational join. Similarly, letting  $K = \mathbb{N}$  we arrive at the bag join operation of SQL. However, as illustrated in the next example, this notion of a join falls short of our expectations.

► **Example 8.** Continuing our running example, the two top tables in Fig. 1 illustrate the projections of the  $\mathbb{T}$ -relation  $\text{Price}(\text{Room}, \text{Date}, \text{Persons})$  on  $\{\text{Room}, \text{Date}\}$  and  $\{\text{Room}, \text{Persons}\}$ . The table in the bottom row is the multiplicative join (4) of the two projections. Note that in the tropical semiring the aforementioned projections are formed as minima of prices, while addition plays the role of multiplication in the join operation. We observe that the multiplicative join is not equivalent to the original price function. In particular, there is no uniform (tropical) scaling factor that returns us  $\text{Price}$  from  $\text{Price}([\text{Room}, \text{Date}]) * \text{Price}([\text{Room}, \text{Persons}])$ .

Two  $K$ -relations  $R(\mathbf{X})$  and  $S(\mathbf{Y})$  are said to be *consistent* if there exists a third relation  $T(\mathbf{XY})$  such that  $T[\mathbf{X}] \equiv R$  and  $T[\mathbf{Y}] \equiv S$ . Atserias and Kolaitis [2] demonstrate that the multiplicative join does not always witness the consistency of two  $K$ -relations, a fact that can be also seen from our running example. Consequently, they introduce a novel join operation which we will now incorporate into our approach. Intuitively this notion of a join is an adaptation of the factorization of a probability distribution obtained from conditional independence. Suppose two random events  $A$  and  $C$  are independent given a third event  $B$ . The joint probability  $P(A, B, C)$  can then be rewritten as  $P(B)P(A \mid B)P(C \mid B) = P(A, B)P(B, C)/P(B)$ . We may recognize that this equation is similar to the multiplicative join of two  $K$ -relations *conditioned* on their common part. In our example this corresponds to multiplying the multiplicative join  $\text{Price}([\text{Room}, \text{Date}]) * \text{Price}([\text{Room}, \text{Persons}])$  with the (tropical) multiplicative inverse of  $\text{Price}([\text{Room}])$ . We observe from Fig. 1 that this sequence of operations yields the initial price function depicted in Tab. 1 (even without re-scaling), in accordance with our expectations.

## 20:8 Conditional Independence on Semiring Relations

Price[Room, Date]		
Room	Date	Price
double	2023-12-01	100
double	2023-08-20	120
twin	2023-08-20	110

Price[Room, Persons]		
Room	Persons	Price
double	1	100
double	2	120
twin	1	110
twin	2	120

Price([Room, Date]) * Price([Room, Persons])			
Room	Date	Persons	Price
double	2023-12-01	1	200
double	2023-12-01	2	220
double	2023-08-20	1	220
double	2023-08-20	2	240
twin	2023-08-20	1	220
twin	2023-08-20	2	230

(Price[Room]) <sup>-1</sup>	
Room	Price
double	-100
twin	-110

■ **Figure 1** Decomposition of the price function.

We will now provide a precise definition of the join operation introduced in [2]. This definition matches the above intuitive description with one exception: Semirings generally lack multiplicative inverses, and therefore the conditioning on the common part of two  $K$ -relations is defined indirectly. For a  $K$ -relation  $R(\mathbf{X})$ , a subset  $\mathbf{Z} \subseteq \mathbf{X}$ , and a  $\mathbf{Z}$ -tuple  $u$ , define

$$c_{R,\mathbf{Z}}^* := \bigotimes_{v \in R[\mathbf{Z}]} R(v) \quad \text{and} \quad c_R(u) := \bigotimes_{\substack{v \in R[\mathbf{Z}] \\ v \neq u}} R(v),$$

with the convention that the empty product evaluates to 1, the neutral element of multiplication in  $K$ . We isolate the following simple property which is applied frequently in the sequel.

► **Proposition 9.** *Suppose  $K$  does not have divisors of zero. If  $R(\mathbf{X})$  is a  $K$ -relation,  $\mathbf{Z} \subseteq \mathbf{X}$ , and  $u$  is a  $\mathbf{Z}$ -tuple, then  $c_{R,\mathbf{Z}}^* \neq 0$  and  $c_R(u) \neq \emptyset$*

If  $R(\mathbf{X})$  and  $S(\mathbf{Y})$  are two  $K$ -relations, the *join*  $R \bowtie S$  of  $R$  and  $S$  is the  $K$ -relation over  $\mathbf{X}\mathbf{Y}$  defined by

$$(R \bowtie S)(t) := R(t[\mathbf{X}])S(t[\mathbf{Y}])c_S(t[\mathbf{X} \cap \mathbf{Y}]). \quad (5)$$

If  $K$  is a semifield (i.e., it has multiplicative inverses), we may rewrite the join as

$$(R \bowtie S)(t) = \frac{c_{S,\mathbf{X} \cap \mathbf{Y}}^* R(t[\mathbf{X}])S(t[\mathbf{Y}])}{S(t[\mathbf{X} \cap \mathbf{Y}])}.$$

The definition of  $R \bowtie S$  is not symmetric, and hence there may be occasions where commutativity fails, i.e.,  $R \bowtie S \neq S \bowtie R$ . However, whenever  $R$  and  $S$  agree on the marginals on their shared variable set  $\mathbf{X} \cap \mathbf{Y}$ , commutativity holds by definition. In particular, Lemma 4 entails that the join of two projections  $R[\mathbf{X}]$  and  $R[\mathbf{Y}]$  of the same relation  $R$  is commutative.

The join operation (5) can also be described in terms of conditional independence and consistency. Suppose  $K$  is a semifield, and suppose  $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$  are pairwise disjoint. Let  $S(\mathbf{X}\mathbf{Y})$  and  $T(\mathbf{X}\mathbf{Z})$  be two normal  $K$ -relations that are consistent. Since equivalence entails identity for normal  $K$ -relations over semifields  $K$ , this is tantamount to finding a normal  $K$ -relation  $R(\mathbf{X}\mathbf{Y}\mathbf{Z})$  such that

$$R[\mathbf{X}\mathbf{Y}] = S \text{ and } R[\mathbf{X}\mathbf{Z}] = T. \quad (6)$$

In particular, Lemma 4 and Eq. (6) yield  $S[\mathbf{X}] = T[\mathbf{X}]$ , whereby  $c_{S,\mathbf{X}}^* = c_{T,\mathbf{X}}^*$ . We may now observe that  $R = 1/c_{T,\mathbf{X}}^*(S \bowtie T)$  is the unique  $K$ -relation that satisfies (6) and the CI  $\mathbf{Y} \perp\!\!\!\perp \mathbf{Z} \mid \mathbf{X}$ . In the particular case where  $K = \mathbb{R}_{\geq 0}$  – in which case  $S$  and  $T$  are two consistent probability distributions – we also know that  $R = 1/c_{T,\mathbf{X}}^*(S \bowtie T)$  is the unique probability distribution that satisfies (6) and maximizes the entropy of  $\mathbf{X}\mathbf{Y}\mathbf{Z}$  (or alternatively, the conditional entropy of  $\mathbf{Y}\mathbf{Z}$  given  $\mathbf{X}$ ) [2].

Let  $R$  be a  $K$ -relation over  $\mathbf{X}\mathbf{Y}$ . The *decomposition of  $R$  along  $\mathbf{X}$  and  $\mathbf{Y}$*  consists of its projections  $R[\mathbf{X}]$  and  $R[\mathbf{Y}]$  on  $\mathbf{X}$  and  $\mathbf{Y}$ , respectively. Such a decomposition is called a *lossless-join decomposition* if  $R[\mathbf{X}] \bowtie R[\mathbf{Y}] \equiv R$ . This definition, which appears already in [2], generalizes the definition of a lossless-join decomposition in database relations. It turns out, as we will next show, that if  $K$  is positive and multiplicatively cancellative, conditional independence holds on a  $K$ -relation if and only if the corresponding decomposition is a lossless-join one.

► **Theorem 10** (Lossless-join decomposition). *Let  $K$  be a positive semiring,  $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$  pairwise disjoint sets of variables, and  $R(\mathbf{X}\mathbf{Y}\mathbf{Z})$  a  $K$ -relation. If  $R$  satisfies  $\mathbf{Y} \perp\!\!\!\perp \mathbf{Z} \mid \mathbf{X}$ , then the decomposition of  $R$  along  $\mathbf{X}\mathbf{Y}$  and  $\mathbf{X}\mathbf{Z}$  is a lossless-join one. If  $K$  is additionally multiplicatively cancellative, then the converse direction holds.*

**Proof.** Assume  $R$  satisfies  $\mathbf{Y} \perp\!\!\!\perp \mathbf{Z} \mid \mathbf{X}$ . We need to show that  $R[\mathbf{X}\mathbf{Y}] \bowtie R[\mathbf{X}\mathbf{Z}] \equiv R$ . Let  $t$  be an arbitrary tuple from  $\text{Tup}(\mathbf{X}\mathbf{Y}\mathbf{Z})$ . By assumption and Lemma 4 we obtain

$$\begin{aligned} (R[\mathbf{X}\mathbf{Y}] \bowtie R[\mathbf{X}\mathbf{Z}])(t) &= R(t[\mathbf{X}\mathbf{Y}])R(t[\mathbf{X}\mathbf{Z}])c_{R[\mathbf{X}\mathbf{Z}]}(t[\mathbf{X}]) \\ &= R(t[\mathbf{X}])R(t) \bigotimes_{\substack{v \in R[\mathbf{X}\mathbf{Z}]' \\ v \neq t[\mathbf{X}]}} R[\mathbf{X}\mathbf{Z}](v) \\ &= R(t[\mathbf{X}])R(t) \bigotimes_{\substack{v \in R[\mathbf{X}]' \\ v \neq t[\mathbf{X}]}} R(v) \\ &= c_{R,\mathbf{X}}^* R(t), \end{aligned}$$

where  $c_{R,\mathbf{X}}^* \neq 0$  by Proposition 9. This proves that  $R[\mathbf{X}\mathbf{Y}] \bowtie R[\mathbf{X}\mathbf{Z}] \equiv R$ .

For the converse direction, suppose  $R[\mathbf{X}\mathbf{Y}] \bowtie R[\mathbf{X}\mathbf{Z}] \equiv R$ . Let  $a, b \in K \setminus \{0\}$  be such that  $aR = b(R[\mathbf{X}\mathbf{Y}] \bowtie R[\mathbf{X}\mathbf{Z}])$ . By Lemma 1, we may assume without loss of generality that  $K$  is a submodel of some positive semifield  $F$ . Hence  $R[\mathbf{X}\mathbf{Y}] \bowtie R[\mathbf{X}\mathbf{Z}] = cR$  for  $c = ab^{-1} \in F$ . We claim that  $c = c_{R,\mathbf{X}}^*$ . Since we assume a non-empty support for each  $K$ -relation, we may select a tuple  $t$  from  $R'$ . By Lemma 4 we have  $t[\mathbf{X}] \in R[\mathbf{X}]'$ , i.e.,  $R(t[\mathbf{X}]) \neq 0$ . We can also deduce the following:

## 20:10 Conditional Independence on Semiring Relations

$$\begin{aligned}
cR(t[\mathbf{X}]) &= \bigoplus_{\substack{t' \in \text{Dup}(\mathbf{XYZ}) \\ t'[\mathbf{X}] = t[\mathbf{X}]}} cR(t') = \bigoplus_{\substack{t' \in \text{Dup}(\mathbf{XYZ}) \\ t'[\mathbf{X}] = t[\mathbf{X}]}} (R[\mathbf{XY}] \bowtie R[\mathbf{XZ}])(t') \\
&= \bigoplus_{\substack{t' \in \text{Dup}(\mathbf{XYZ}) \\ t'[\mathbf{X}] = t[\mathbf{X}]}} R(t'[\mathbf{XY}])R(t'[\mathbf{XZ}])c_{R[\mathbf{XZ}]}(t'[\mathbf{X}]) \\
&= c_{R[\mathbf{XZ}]}(t'[\mathbf{X}]) \bigoplus_{\substack{t' \in \text{Dup}(\mathbf{XY}) \\ t'[\mathbf{X}] = t[\mathbf{X}]}} R(t'[\mathbf{XY}]) \bigoplus_{\substack{t' \in \text{Dup}(\mathbf{XZ}) \\ t'[\mathbf{X}] = t[\mathbf{X}]}} R(t'[\mathbf{XZ}]) \\
&= R(t[\mathbf{X}])R(t[\mathbf{X}]) \bigotimes_{\substack{v \in R[\mathbf{X}]' \\ v \neq t[\mathbf{X}]}} R(v) = c_{R, \mathbf{X}}^* R(t[\mathbf{X}]).
\end{aligned}$$

Multiplying (in  $F$ ) by the inverse of  $R(t[\mathbf{X}])$  then yields  $c = c_{R, \mathbf{X}}^*$ , proving our claim.

Since  $R[\mathbf{XY}] \bowtie R[\mathbf{XZ}] = c_{R, \mathbf{X}}^* R$ , we may apply the sequence of equations from the previous case to obtain that for all  $t \in \text{Dup}(\mathbf{XYZ})$ ,

$$R(t[\mathbf{XY}])R(t[\mathbf{XZ}])c_{R[\mathbf{XZ}]} = R(t[\mathbf{X}])R(t)c_{R[\mathbf{XZ}]}.$$

Since  $c_{R[\mathbf{XZ}]}$  is non-zero by Proposition 9, it can be removed from both sides of the equation by multiplicative cancellativity. We conclude that  $R$  satisfies  $\mathbf{Y} \perp\!\!\!\perp \mathbf{Z} \mid \mathbf{X}$ .  $\blacktriangleleft$

The preceding proof entails that over positive and multiplicatively cancellative semirings  $K$ , the satisfaction of  $\mathbf{Y} \perp\!\!\!\perp \mathbf{Z} \mid \mathbf{X}$  by a  $K$ -relation  $R(\mathbf{XYZ})$  holds if and only if  $R[\mathbf{XY}] \bowtie R[\mathbf{XZ}] = c_{R, \mathbf{X}}^* R$ . If  $K$  is additionally a semifield, then  $R \models \mathbf{Y} \perp\!\!\!\perp \mathbf{Z} \mid \mathbf{X}$  exactly when we find two  $K$ -relations  $S(\mathbf{XY})$  and  $T(\mathbf{XZ})$  such that  $R(t) = S(t[\mathbf{XY}])T(t[\mathbf{XZ}])$ .

Our running example demonstrates that semiring interpretations can give rise to lossless-join decompositions which are unattainable under the relational interpretation.

► **Example 11.** Consider again Tab. 1 as a  $\mathbb{T}$ -relation  $\text{Price}(\text{Room}, \text{Date}, \text{Persons})$ . As we see from Fig. 1, this  $\mathbb{T}$ -relation decomposes along  $\{\text{Room}, \text{Date}\}$  and  $\{\text{Room}, \text{Persons}\}$ . In particular,  $\text{Price}[\text{Room}, \text{Date}] \bowtie \text{Price}[\text{Room}, \text{Persons}] = c_{\text{Price}, \{\text{Room}\}}^* \text{Price}$  where  $c_{\text{Price}, \{\text{Room}\}}^* = 210$ . However, viewed as an ordinary relation  $R$  over  $\{\text{Price}, \text{Room}, \text{Date}, \text{Persons}\}$  this table is in sixth normal form, meaning that no decomposition along  $X_1, \dots, X_n$  is a lossless-join one, unless  $X_i$  for some  $i$  is the full variable set. Specifically, for any  $X \subsetneq \{\text{Price}, \text{Room}, \text{Date}, \text{Persons}\}$ , the projection of the tuple (double, 2023-08-20, 2, 120) on  $X$  is in  $R[X]$ , even though the tuple itself does not belong to  $R$ .

Examples of positive semirings which are not multiplicatively cancellative seem somewhat artificial. Consider  $K = (\mathbb{N}_{>0}, \mathbb{Z}_2) \cup \{(0, 0)\}$  with the semiring structure pointwise inherited from  $\mathbb{N}$  and  $\mathbb{Z}_2$ . Note that  $K$  is positive but violates multiplicative cancellativity, as  $(1, 1) \otimes (1, 0) = (1, 0) \otimes (1, 0)$ , while  $(1, 1) \neq (1, 0) \neq (0, 0)$ . Using  $K$  we can demonstrate that the assumption of multiplicative cancellativity cannot be dropped from the second statement of Lemma 10. We write  $\mathbf{X} \perp\!\!\!\perp \mathbf{Y}$  for the *marginal independence* between  $\mathbf{X}$  and  $\mathbf{Y}$ , defined as the CI  $\mathbf{X} \perp\!\!\!\perp \mathbf{Y} \mid \emptyset$ . Consider the  $K$ -relation  $R$  from Fig. 2. This  $K$ -relation does not satisfy  $A \perp\!\!\!\perp B$ : Choosing  $t(A, B) = (0, 0)$  we observe  $R(t[A]) \otimes R(t[B]) = (2, 1) \otimes (2, 1) \neq (4, 0) \otimes (1, 0) = R(\emptyset) \otimes R(t[AB])$ . On the other hand, we have  $R \equiv R[A] \bowtie R[B]$  because  $aR = b(R[A] \bowtie R[B])$ , where  $a = (4, 0) \neq (0, 0) \neq (1, 0) = b$ .

Similarly, the assumption of positivity is necessary for the first statement of Lemma 10. Suppose  $a, b \in K \setminus \{0\}$  are such that  $a \oplus b = 0$ , and consider variables  $X, Y$  with domain  $\{0, 1\}$ .

$R$		
$A$	$B$	$\#$
0	0	(1,0)
0	1	(1,1)
1	0	(1,1)
1	1	(1,0)

$R[C], C \in \{A, B\}$	
$C$	$\#$
0	(2,1)
1	(2,1)

$R[A] \bowtie R[B]$		
$A$	$B$	$\#$
0	0	(4,1)
0	1	(4,1)
1	0	(4,1)
1	1	(4,1)

■ **Figure 2** Decomposition without independence.

- (S1) Triviality:  $Y \perp\!\!\!\perp \emptyset \mid X$ .
- (S2) Symmetry:  $Y \perp\!\!\!\perp Z \mid X$ , then  $Z \perp\!\!\!\perp Y \mid X$ .
- (S3) Decomposition:  $Y \perp\!\!\!\perp ZW \mid X$ , then  $Y \perp\!\!\!\perp Z \mid X$ .
- (S4) Weak union:  $Y \perp\!\!\!\perp ZW \mid X$ , then  $Y \perp\!\!\!\perp W \mid XZ$ .
- (S5) Contraction:  $Y \perp\!\!\!\perp Z \mid X$  and  $Y \perp\!\!\!\perp W \mid XZ$ , then  $Y \perp\!\!\!\perp ZW \mid X$ .
- (G) Interaction:  $Y \perp\!\!\!\perp Z \mid XW$  and  $Y \perp\!\!\!\perp W \mid XZ$ , then  $Y \perp\!\!\!\perp ZW \mid X$ .

■ **Figure 3** Semigraphoid axioms (S1–S5) and graphoid axioms (S1–S5,G).

Then, the  $K$ -relation  $R(XY)$  corresponding to the set  $\{(0,0;a), (0,1;b), (1,0;b), (1,1;a)\}$  of triples  $(t(X), t(Y); R(t))$  satisfies  $X \perp\!\!\!\perp Y$ , but the decomposition along  $X$  and  $Y$  is obviously not a lossless-join one. In fact, the definition of the marginal, Eq. (2), may not even be useful if  $K$  is not positive. For instance, a pure quantum state  $|\psi\rangle_{XY}$  within a finite-dimensional composite Hilbert space  $\mathcal{H}_{XY}$  can be conceived as a  $\mathbb{C}$ -relation  $R(XY)$  over complex numbers  $\mathbb{C}$ . Its marginal with respect to  $\mathcal{H}_X$  is however not obtained from Eq. (2), but through a partial trace of the relevant density matrix. The marginal state may not even be a  $\mathbb{C}$ -relation anymore, because it can be mixed, i.e., a probability distribution over pure states.

## 5 Axiomatic properties

The previous section identifies positivity and multiplicative cancellativity as the key semiring properties underlying the correspondence between conditional independence and lossless-join decompositions. The main observation of the present section will be that the same key semiring properties guarantee soundness and completeness of central axiomatic properties associated with CIs.

### 5.1 Semigraphoid axioms

The *semigraphoid axioms* [21] (the first five rules in Fig. 3) are a collection of fundamental conditional independence properties observed in various contexts, including database relations and probability distributions. The *graphoid axioms* are obtained by extending the semigraphoid axioms with the interaction rule (the last rule in Fig. 3). While not sound in general, the interaction rule is known to hold for probability distributions in which every probability is positive. We observe next that these results extend to  $K$ -relations whenever  $K$  is positive and multiplicatively cancellative; the interaction rule, in particular, is sound over total  $K$ -relations.

To offer context for Theorem 12 which is proven in Appendix B, recall from information theory the concept of *conditional mutual information*, which can be defined over sets of random variables  $U, V, W$  as  $I(V; W \mid U) := H(UV) + H(UW) - H(U) - H(UVW)$ ,

## 20:12 Conditional Independence on Semiring Relations

where  $H$  is the Shannon entropy. The conditional mutual information  $I(\mathbf{V}; \mathbf{W} \mid \mathbf{U})$  is zero if and only if the CI  $\mathbf{V} \perp\!\!\!\perp \mathbf{W} \mid \mathbf{U}$  holds in the underlying probability distribution. Now, consider the *chain rule*

$$I(\mathbf{Y}; \mathbf{Z} \mid \mathbf{X}) + I(\mathbf{Y}; \mathbf{W} \mid \mathbf{XZ}) = I(\mathbf{Y}; \mathbf{ZW} \mid \mathbf{X})$$

of conditional mutual information. Since conditional mutual information is non-negative, the chain rule readily entails Decomposition, Weak union, and Contraction for probability distributions. In the semiring setting we cannot deduce these rules analogously, as there seems to be no general measure to capture conditional independence over  $K$ -relations. We can however use the measure-theoretic interpretation of conditional independence as a guide toward a proof. Consider, for instance, the contraction rule, which can be restated in the information context as follows: if

$$H(\mathbf{XY}) + H(\mathbf{XZ}) = H(\mathbf{X}) + H(\mathbf{XYZ}) \quad \text{and} \quad (7)$$

$$H(\mathbf{XYZ}) + H(\mathbf{XZW}) = H(\mathbf{XZ}) + H(\mathbf{XYZW}), \quad (8)$$

then

$$H(\mathbf{XY}) + H(\mathbf{XZW}) = H(\mathbf{X}) + H(\mathbf{XYZW}). \quad (9)$$

In particular, Eq. (9) is a consequence of subtracting  $H(\mathbf{XZ}) + H(\mathbf{XYZ})$  from the combination of Eqs. (7) and (8). The soundness proof for  $K$ -relations has now the same general structure. Instantiations of Eq. (3) for the CIs appearing in the contraction rule are structurally similar to Eqs. (7), (8), and (9), with addition between entropies being replaced by multiplication within  $K$ . Instead of subtraction, one now applies multiplicative cancellativity to remove all superfluous terms from the combination of two equations. Additionally, one has to deal with those cases where the terms to be eliminated are zero, and multiplicative cancellativity cannot be applied.

► **Theorem 12.** *Triviality, Symmetry, and Decomposition are sound for  $K$ -relations. Weak union and Contraction are sound for  $K$ -relations where  $K$  is positive and multiplicatively cancellative. Interaction is sound for total  $K$ -relations where  $K$  is positive and multiplicatively cancellative.*

Having considered the graphoid axioms for  $K$ -relations, we next consider the interaction between conditional independence and functional dependence.

### 5.2 Functional dependencies

Given two sets of variables  $\mathbf{X}$  and  $\mathbf{Y}$ , the expression  $\mathbf{X} \rightarrow \mathbf{Y}$  is called a *functional dependency* (FD). A relation  $R$  satisfies  $\mathbf{X} \rightarrow \mathbf{Y}$ , denoted  $R \models \mathbf{X} \rightarrow \mathbf{Y}$ , if for all  $t, t' \in R$ ,  $t[\mathbf{X}] = t'[\mathbf{X}]$  implies  $t[\mathbf{Y}] = t'[\mathbf{Y}]$ . We extend this definition to  $K$ -relations  $R$  by stipulating that  $R$  satisfies an FD  $\sigma$  whenever its support  $R'$  satisfies  $\sigma$ .

The Armstrong axioms for FDs [1] comprise the first three rules in Fig. 4. These rules are sound and complete for database relations, and hence, by definition, for  $K$ -relations over any  $K$ . The last two rules are two combination rules for MVDs and FDs [4] rewritten in different syntax. To extend these rules  $K$ -relations, we again need positivity and multiplicative cancellativity. The following proposition is proven in the arXiv version [13].

► **Proposition 13.** *CI introduction is sound for all  $K$ -relations, where  $K$  is  $\oplus$ -positive. FD contraction is sound for all  $K$ -relations, where  $K$  is positive and multiplicatively cancellative.*

- (FD1) Triviality: if  $Y \subseteq X$ , then  $X \rightarrow Y$ .
- (FD2) Augmentation: if  $X \rightarrow Y$  and  $XZ \rightarrow YZ$ .
- (FD3) Transitivity: if  $X \rightarrow Y$  and  $Y \rightarrow Z$ , then  $X \rightarrow Z$ .
- (FD-CI1) CI introduction: if  $X \rightarrow Y$ , then  $Y \perp\!\!\!\perp Z \mid X$ .
- (FD-CI2) FD contraction: if  $Y \perp\!\!\!\perp Z \mid X$  and  $XY \rightarrow Z$ , then  $X \rightarrow Z$ .

■ **Figure 4** Armstrong's axioms (FD1–FD3) and combination rules (FD–CI1,FD–CI2).

Soundness of CI introduction means that, for positive  $K$ , a functional dependency on a  $K$ -relation leads to a lossless-join decomposition. The next proposition, stating this fact, was proven originally in [2]. Alternatively, we now see that the proposition follows directly by Theorem 10 and Proposition 13.

► **Proposition 14** ([2]). *Let  $K$  be a positive semiring,  $X, Y, Z$  be pairwise disjoint sets of variables, and  $R(XYZ)$  be a  $K$ -relation. If  $R$  satisfies  $X \rightarrow Y$ , then the decomposition of  $R$  along  $XY$  and  $XZ$  is a lossless-join one.*

We have now examined fundamental inference rules for CIs and FDs that have their origins in database theory and probability theory. The combination of these rules however is not – and cannot be – complete in either context. Specifically, over both finite relations and finite distributions, the implication problems for EMVDs/CIs are not even r.e. since the problems are known to be undecidable [14, 18, 20] and co-r.e. [17]. In the next section we restrict attention to saturated CIs which are known to exhibit favorable algorithmic and axiomatic properties.

### 5.3 Saturated conditional independence and functional dependence

We next show that SCI+FD enjoys a complete axiomatization that is shared by all positive and multiplicatively cancellative semirings  $K$ . This result readily entails that logical implication within the class SCI+FD does not depend on the chosen semiring  $K$ , provided that it has the fundamental properties mentioned above.

Given a set  $\Sigma \cup \{\tau\}$  of dependencies, we say that  $\Sigma$  *implies*  $\tau$  over relations (resp.  $K$ -relations), denoted  $\Sigma \models \tau$  (resp.  $\Sigma \models_K \tau$ ), if every relation (resp.  $K$ -relation) satisfying  $\Sigma$  satisfies  $\tau$ . Let  $\sigma \mapsto \sigma^*$  associate an SCI/CI with its corresponding MVD/EMVD. Extend this mapping to be the identity on FDs, and extend it to sets in the natural way:  $\Sigma^* = \{\sigma^* \mid \sigma \in \Sigma\}$ .

► **Theorem 15.** *Let  $K$  be a positive and multiplicatively cancellative semiring. Let  $\Sigma \cup \{\tau\}$  be a set of SCIs and FDs. The following are equivalent:*

1.  $\tau$  can be derived from  $\Sigma$  using (S1–S5), (FD1–FD3), and (FD–CI1,FD–CI2).
2.  $\Sigma$  *implies*  $\tau$  over  $K$ -relations.
3.  $\Sigma^*$  *implies*  $\tau^*$  over relations consisting of two tuples.
4.  $\Sigma^*$  *implies*  $\tau^*$  over relations.

**Proof.** (1)  $\Rightarrow$  (2). This direction is immediate due to Theorem 12, Proposition 13, and soundness of the Armstrong axioms for ordinary relations. (2)  $\Rightarrow$  (3). Any two-tuple relation  $R = \{t, t'\}$  can be transformed to a  $K$ -relation  $S$  such that the support  $S'$  is  $R$ , and  $S(t) = S'(t) = 1$ . It is straightforward to verify that  $R$  satisfies  $\sigma^*$  if and only if  $S$  satisfies  $\sigma$ , for all CIs and FDs  $\sigma$ . From this, the direction follows. (3)  $\Rightarrow$  (4). This direction has



## 20:14 Conditional Independence on Semiring Relations

been proven in [22]. (4)  $\Rightarrow$  (1). This direction follows from the fact that the system (S1-S5), (FD1-FD3), (FD-CI1,FD-CI2) mirrors the complete axiomatization of MVDs and FDs. We give an explicit proof in the arXiv version [13].  $\blacktriangleleft$

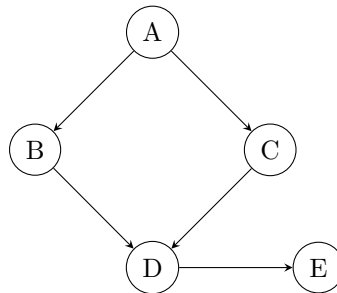
► **Corollary 16.** *Let  $K, K'$  be positive and multiplicatively cancellative semirings, and let  $\Sigma \cup \{\tau\}$  be a set of SCIs and FDs. Then,  $\Sigma$  implies  $\tau$  over  $K$ -relations if and only if  $\Sigma$  implies  $\tau$  over  $K'$ -relations.*

Theorem 10 and Corollary 16 demonstrate that the decomposition properties arising from multivalued and functional dependencies hold invariably for all  $K$ -relations, given  $K$  is multiplicatively cancellative and positive. Standard database normalization methods thus extend to diverse contexts and may sometimes coincide with existing methods. The following example shows that relational normalizations can sometimes match the factorizations of probability distributions arising from Bayesian networks.

A relational database schema is a set of relation schemata, each associated with a set of constraints. It is in *fourth normal form* (4NF) if for any of its MVD constraints  $\mathbf{X} \twoheadrightarrow \mathbf{Y}$ ,  $\mathbf{X}$  is a superset of a key. A *Bayesian network* is a directed acyclic graph in which the nodes represent random variables and the directed edges probabilistic dependencies between variables. Each node is thus directly influenced by its parents in the graph. Conversely, each node indirectly influences its descendants by transitivity. The *local Markov property* states that once the parent nodes are known, the state of the current node does not reveal any additional information about the states of its non-descendants, i.e., each node is conditionally independent of its non-descendants given its parents.

► **Example 17.** Consider the Bayesian network in Fig. 5. The chain rule of probability distributions and the local Markov property implies that the joint distribution  $P(A, B, C, D, E)$  has a factorization  $P(A)P(B | A)P(C | A)P(D | BC)P(E | D)$ .

The local Markov property produces three non-trivial CIs up to symmetry, rewritten as the following EMVDs  $A \twoheadrightarrow B|C$ ,  $BC \twoheadrightarrow A|D$ ,  $D \twoheadrightarrow ABC|E$ . Suppose our goal is to transform the unirelational database schema  $\{ABCDE\}$  into 4NF, assuming absence of key constraints. Since the last EMVD is also an MVD, we first decompose  $ABCDE$  along  $ABCD$  and  $DE$ . Since the second EMVD is an MVD on  $ABCD$ , we continue by splitting  $ABCD$  into  $ABC$  and  $BCD$ . To remove the last remaining MVD, we decompose  $ABC$  along  $AB$  and  $AC$ . The final schema  $\{AB, AC, BCD, DE\}$  is free of MVDs, and thus in 4NF. Furthermore, the decomposition of  $P$  (as a  $\mathbb{R}_{\geq 0}$ -relation) along  $\{AB, AC, BCD, DE\}$  reproduces the aforementioned factorization of  $P$  into conditional probabilities.



■ **Figure 5** A simple Bayesian network.

## 6 Comparison of implication

As mentioned previously, implication for non-saturated CIs depends heavily on the underlying semantics. In this section, we examine the connections between different conditional independence semantics in relation to the semiring properties they rely on. Using model-theoretic arguments, we first show that  $\Sigma \models_{\mathbb{R}_{\geq 0}} \tau$  implies  $\Sigma \models_K \tau$ , whenever  $K$  is cancellative and equipped with a natural total order.

Consider a CI of the form  $\tau = \mathbf{Y} \perp\!\!\!\perp \mathbf{Z} \mid \mathbf{X}$ , where  $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$  are disjoint subsets of a schema  $\mathbf{V}$ . Suppose the domain of each variable in  $\mathbf{V}$  is finite. For each  $\mathbf{V}$ -tuple  $t$ , introduce a variable  $x_t$ . Denote by  $\vec{x}_{\mathbf{V}}$  a sequence listing all variables  $x_t$ ,  $t \in \text{Tup}(\mathbf{V})$ . We associate  $\tau$  and  $\mathbf{V}$  with a quantifier-free first-order arithmetic formula

$$\phi_{\tau, \mathbf{V}} := \bigwedge_{\substack{t_{\mathbf{X}} \in \text{Tup}(\mathbf{X}) \\ t_{\mathbf{Y}} \in \text{Tup}(\mathbf{Y}) \\ t_{\mathbf{Z}} \in \text{Tup}(\mathbf{Z})}} \bigoplus_{t \in \text{Tup}(\mathbf{X}\mathbf{Y}\mathbf{Z})} x_t \bigoplus_{\substack{t \in \text{Tup}(\mathbf{X}\mathbf{Y}\mathbf{Z}) \\ t[\mathbf{X}] = t_{\mathbf{X}} \\ t[\mathbf{Y}] = t_{\mathbf{Y}} \\ t[\mathbf{Z}] = t_{\mathbf{Z}}}} x_t = \bigoplus_{t \in \text{Tup}(\mathbf{X}\mathbf{Y}\mathbf{Z})} x_t \bigoplus_{\substack{t \in \text{Tup}(\mathbf{X}\mathbf{Y}\mathbf{Z}) \\ t[\mathbf{X}] = t_{\mathbf{X}} \\ t[\mathbf{Y}] = t_{\mathbf{Y}} \\ t[\mathbf{Z}] = t_{\mathbf{Z}}}} x_t.$$

A formula  $\phi$  is said to be *universal* if it is of the form  $\forall x_1 \dots \forall x_n \theta$ , where  $\theta$  is quantifier-free. All universal first-order properties of a model are preserved for its submodels (and any of their isomorphic copies) [6].

► **Proposition 18.** *Let  $\mathcal{A}$  and  $\mathcal{B}$  be models over a vocabulary  $\tau$ , and let  $\phi$  be a universal first-order sentence over  $\tau$ . If  $\mathcal{A}$  embeds in  $\mathcal{B}$ , then  $\mathcal{A} \models \phi$  implies  $\mathcal{B} \models \phi$ .*

The following theorem lists some basic properties of real-closed fields [5, 6]. A field  $F$  is called *real* if it can be associated with an ordering  $\leq$  such that  $(F, \leq)$  becomes an ordered field. A *field*  $F'$  is an *extension* of a field  $F$  if  $F \subseteq F'$ , and the field operations of  $F$  are those inherited from  $F'$ . The extension is *proper* if  $F$  is a strict subset of  $F'$ , and *algebraic* if every element in  $F'$  is a root of a non-zero polynomial with coefficients in  $F$ . A real field with no proper real algebraic extension is called *real closed*. For instance, the field of real numbers is real closed, whereas the field of rational numbers is real but not real closed. On the other hand, no finite or algebraically closed field is real. A real closed field has a unique ordering, which is definable by  $a \leq b : \Leftrightarrow \exists c(a \oplus c^2 = b)$ . An algebraic extension  $F'$  of an ordered field  $(F, \leq)$  is called a *real closure* of  $F$  if  $F'$  is real closed, and its unique ordering extends that of  $F$  (i.e., the ordering is preserved under the inclusion map  $F \hookrightarrow F'$ ). Two models  $\mathcal{A}$  and  $\mathcal{B}$  are *elementarily equivalent*, written  $\mathcal{A} \equiv \mathcal{B}$ , if they satisfy the same first-order sentences.

► **Theorem 19.**

- Any totally ordered field  $(F, \leq)$  has a real closure  $F'$ .
- If  $(F, \leq)$  is a totally ordered field, and  $F_0$  and  $F_1$  are its real closures uniquely ordered by  $\leq_0$  and  $\leq_1$ , there is an isomorphism between  $(F_0, \leq_0)$  and  $(F_1, \leq_1)$  which is identity on  $F$ .
- Any two real-closed fields  $F_0$  and  $F_1$  are elementarily equivalent.

We can now prove the property that  $\mathbb{R}_{\geq 0}$ -implication entails  $K$ -implication, for any semiring  $K$  embedded in a cancellative and naturally totally ordered one.

► **Theorem 20.** *Let  $\Sigma \cup \{\tau\}$  be a finite set of CIs, and suppose  $K$  embeds in a naturally totally ordered cancellative semiring. Then,  $\Sigma \models_{\mathbb{R}_{\geq 0}} \tau$  implies  $\Sigma \models_K \tau$ .*

**Proof.** By Lemma 2, Theorem 19, and transitivity of the embedding relation,  $K$  embeds in a real-closed field  $F$ . Let  $R(\mathbf{V})$  be a  $K$ -relation, where  $\mathbf{V}$  is a set of variables that includes each variable appearing in  $\Sigma \cup \{\tau\}$ . We need to show that  $R \models \Sigma$  implies  $R \models \tau$ . Since

## 20:16 Conditional Independence on Semiring Relations

satisfaction of a CI by  $R$  does not depend on tuple values that do not appear in  $R$ , we may without loss of generality assume that the domain of each variable in  $\mathbf{V}$  is finite. Then, assuming  $\Sigma = \{\sigma_1, \dots, \sigma_n\}$ , we may consider the universal first-order sentence

$$\psi_{\Sigma, \tau, \mathbf{V}} := \forall \vec{x}_{\mathbf{V}} (\vec{x}_{\mathbf{V}} \geq \vec{0} \wedge \phi_{\sigma_1, \mathbf{V}} \wedge \dots \wedge \phi_{\sigma_n, \mathbf{V}} \rightarrow \phi_{\tau, \mathbf{V}}),$$

where, given a sequence  $\vec{x} = (x_1, \dots, x_l)$ , we write  $\vec{x} \geq \vec{0}$  as a shorthand for  $x_1 \geq 0 \wedge \dots \wedge x_l \geq 0$ . Since  $\Sigma \models_{\mathbb{R}_{\geq 0}} \tau$  by hypothesis,  $\psi_{\Sigma, \tau, \mathbf{V}}$  must be true for the field of reals  $\mathbb{R}$  (equipped with its unique ordering). Since  $F$  and  $\mathbb{R}$  are real-closed, they share the same first-order properties; in particular,  $F$  also satisfies  $\psi_{\Sigma, \tau, \mathbf{V}}$ . By Proposition 18,  $K$  likewise satisfies  $\psi_{\Sigma, \tau, \mathbf{V}}$ , and hence  $R \models \Sigma$  implies  $R \models \tau$ . We conclude that  $\Sigma \models_K \tau$ .  $\blacktriangleleft$

The preceding theorem readily entails that implication over  $\mathbb{R}_{\geq 0}$  entails implication over  $\mathbb{N}_{\geq 0}$  and  $\mathbb{Q}_{\geq 0}$ . Another example is  $K = \mathbb{N} \times \mathbb{N}$  with pointwise addition and multiplication, and neutral elements  $(0, 0)$  and  $(1, 1)$ . This semiring is not naturally totally ordered, because it contains incomparable elements, such as  $(0, 1)$  and  $(1, 0)$ . However, it can be extended to  $K \cup (\mathbb{Z} \times \mathbb{N}_{>0})$  which is naturally totally ordered and cancellative. For another example, consider the semiring  $K = \mathbb{N}[X]$  of polynomials in  $X$  with coefficients from natural numbers. Since  $K$  contains incomparable elements, such as  $X + 2$  and  $2X + 1$ , its natural order is not total. The extension of  $K$  with those polynomials of  $\mathbb{Z}[X]$  in which the leading coefficient is positive produces a cancellative semiring whose natural order is total.

Let us then turn attention to the Boolean semiring  $\mathbb{B}$  and its connections with other semirings. First we note that although both  $\mathbb{R}_{\geq 0}$  and  $\mathbb{B}$  are naturally totally ordered, only the first one is additively cancellative. In light of Theorem 20, this may help explain why there is no implication from  $\Sigma \models_{\mathbb{R}_{\geq 0}} \tau$  to  $\Sigma \models_{\mathbb{B}} \tau$ . Another difference is that only the Boolean semiring is associated with an idempotent addition; an operation  $*$  on  $K$  is said to be *idempotent* if  $a * a = a$  for all  $a \in K$ . We observe that  $\mathbb{B}$ -implication entails  $K$ -implication, whenever  $K$  has an idempotent addition.

► **Proposition 21.** *Let  $K$  be a semiring associated with an idempotent addition. Let  $\Sigma \cup \{\tau\}$  be a set of CIs. Then,  $\Sigma \models_K \tau$  implies  $\Sigma \models_{\mathbb{B}} \tau$ .*

**Proof.** Recall that we consider only non-trivial semirings  $K$ , where  $0 \neq 1$ . Thus, any  $\mathbb{B}$ -relation  $R$  can be readily interpreted as a  $K$ -relation  $R'$ . The idempotence of addition guarantees that  $R \models \sigma$  if and only if  $R' \models \sigma$ , for any CI  $\sigma$ . The statement of the lemma then follows.  $\blacktriangleleft$

We leave it as an open question whether the statements of Theorem 20 and Proposition 21 hold also in the converse directions.

## 7 Conclusion

We have studied axiomatic and decomposition properties of conditional independence over  $K$ -relations. For positive and multiplicatively cancellative  $K$ , we showed that (i) conditional independence corresponds to lossless-join decompositions, (ii) the semigraphoid axioms of conditional independence are sound, and (iii) saturated conditional independence and functional dependence have a sound and complete axiom system, mirroring the sound and complete axiom system of MVDs and FDs. To demonstrate possible applications, we provided an example data table that admits a lossless-join decomposition only when one of its variables is reinterpreted as a semiring annotation. Finally, we considered a model-theoretic approach to study the relationships between different CI semantics.

The questions of the axiomatic characterization [15, 26, 28] and decidability [14, 20] of the CI implication problem have been answered in the negative in different frameworks. Having identified positivity and multiplicative cancellativity as the fundamental semiring properties for the notion of conditional independence, we may now ask whether these negative results extend to any  $K$  with these characteristics.

---

## References

- 1 William W. Armstrong. Dependency Structures of Data Base Relationships. In *Proc. of IFIP World Computer Congress*, pages 580–583, 1974.
- 2 Albert Atserias and Phokion G. Kolaitis. Consistency, acyclicity, and positive semirings. In Alessandra Palmigiano and Mehrnoosh Sadrzadeh, editors, *Samson Abramsky on Logic and Structure in Computer Science and Beyond*, pages 623–668, Cham, 2023. Springer International Publishing.
- 3 Timon Barlag, Miika Hannula, Juha Kontinen, Nina Pardal, and Jonni Virtema. Unified foundations of team semantics via semirings. In *KR*, pages 75–85, 2023. doi:10.24963/kr.2023/8.
- 4 Catriel Beeri, Ronald Fagin, and John H. Howard. A complete axiomatization for functional and multivalued dependencies in database relations. In *SIGMOD Conference*, pages 47–61. ACM, 1977. doi:10.1145/509404.509414.
- 5 J. Bochnak, M. Coste, and M-F. Roy. *Real Algebraic Geometry*. Springer, 1998.
- 6 Chen C. Chang and H. Jerome Keisler. *Model theory, Third Edition*, volume 73 of *Studies in logic and the foundations of mathematics*. North-Holland, 1992.
- 7 Shumo Chu, Brendan Murphy, Jared Roesch, Alvin Cheung, and Dan Suciu. Axiomatic foundations and algorithms for deciding semantic equivalences of SQL queries. *Proc. VLDB Endow.*, 11(11):1482–1495, 2018. doi:10.14778/3236187.3236200.
- 8 A. P. Dempster. A generalization of bayesian inference. *Journal of the Royal Statistical Society. Series B (Methodological)*, 30(2):205–247, 1968.
- 9 Pietro Galliani and Jouko Väänänen. Diversity, dependence and independence. *Ann. Math. Artif. Intell.*, 90(2-3):211–233, 2022. doi:10.1007/s10472-021-09778-8.
- 10 Dan Geiger, Thomas Verma, and Judea Pearl. Identifying independence in bayesian networks. *Networks*, 20(5):507–534, 1990. doi:10.1002/net.3230200504.
- 11 Todd J. Green, Gregory Karvounarakis, and Val Tannen. Provenance semirings. In *PODS*, pages 31–40. ACM, 2007. doi:10.1145/1265530.1265535.
- 12 Marc Gyssens, Mathias Niepert, and Dirk Van Gucht. On the completeness of the semigraphoid axioms for deriving arbitrary from saturated conditional independence statements. *Inf. Process. Lett.*, 114(11):628–633, 2014. doi:10.1016/j.ip1.2014.05.010.
- 13 Miika Hannula. Conditional independence on semiring relations. *CoRR*, abs/2310.01910, 2023. doi:10.48550/arXiv.2310.01910.
- 14 Christian Herrmann. On the undecidability of implications between embedded multivalued database dependencies. *Information and Computation*, 122(2):221–235, 1995. doi:10.1006/inco.1995.1148.
- 15 Douglas Stott Parker Jr. and Kamran Parsaye-Ghomi. Inferences involving embedded multivalued dependencies and transitive dependencies. In *SIGMOD Conference*, pages 52–57. ACM Press, 1980. doi:10.1145/582250.582259.
- 16 Batya Kenig and Dan Suciu. Integrity constraints revisited: From exact to approximate implication. *Log. Methods Comput. Sci.*, 18(1), 2022. doi:10.46298/LMCS-18(1:5)2022.
- 17 Mahmoud Abo Khamis, Phokion G. Kolaitis, Hung Q. Ngo, and Dan Suciu. Decision problems in information theory. In *ICALP*, volume 168 of *LIPICs*, pages 106:1–106:20, 2020. doi:10.4230/LIPICs.ICALP.2020.106.
- 18 Lukas Kühne and Geva Yashfe. On entropic and almost multilinear representability of matroids. *CoRR*, abs/2206.03465, 2022. doi:10.48550/arXiv.2206.03465.

- 19 Tony T. Lee. An information-theoretic analysis of relational databases - part I: data dependencies and information metric. *IEEE Trans. Software Eng.*, 13(10):1049–1061, 1987. doi:10.1109/TSE.1987.232847.
- 20 Cheuk Ting Li. Undecidability of network coding, conditional information inequalities, and conditional independence implication. *IEEE Trans. Inf. Theory*, 69(6):3493–3510, 2023. doi:10.1109/TIT.2023.3247570.
- 21 J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, CA, 1988.
- 22 Yehoshua Sagiv, Claude Delobel, Douglas Stott Parker Jr., and Ronald Fagin. An equivalence between relational database dependencies and a fragment of propositional logic. *J. ACM*, 28(3):435–453, 1981. doi:10.1145/322261.322263.
- 23 Glenn Shafer. *A Mathematical Theory of Evidence*. Princeton University Press, Princeton, 1976.
- 24 Wolfgang Spohn. Ordinal conditional functions. a dynamic theory of epistemic states. In W. L. Harper and B. Skyrms, editors, *Causation in Decision, Belief Change, and Statistics, vol. II*. Kluwer Academic Publishers, 1988.
- 25 Milan Studený. Multiinformation and the problem of characterization of conditional independence relations. *Problems of Control and Information Theory*, 18(1):3–16, 1989.
- 26 Milan Studený. Conditional independence relations have no finite complete characterization. *Transactions of the 11th Prague Conference on Information Theory*, pages 377–396, 1992.
- 27 Milan Studený. Formal properties of conditional independence in different calculi of AI. In *ECSQARU*, volume 747 of *Lecture Notes in Computer Science*, pages 341–348. Springer, 1993. doi:10.1007/BFb0028219.
- 28 Milan Studený. Conditional independence and natural conditional functions. *Int. J. Approx. Reason.*, 12(1):43–68, 1995. doi:10.1016/0888-613X(94)00014-T.
- 29 L.A Zadeh. Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*, 1(1):3–28, 1978.

## A Embeddings

Let  $\tau$  be a first-order vocabulary consisting of function and relation symbols (constant symbols can be viewed as 0-ary function symbols). We write  $\text{ar}(\alpha)$  for the *arity* of a symbol  $\alpha \in \tau$ . Given a  $\tau$ -structure  $\mathcal{M}$  and an element  $\alpha$  from  $\tau$ , we write  $\alpha^{\mathcal{M}}$  for the interpretation of  $\alpha$  in  $\mathcal{M}$ . Let  $\mathcal{A}$  and  $\mathcal{B}$  be two  $\tau$ -structures with domains  $A$  and  $B$ . We call  $\mathcal{A}$  a *submodel* of  $\mathcal{B}$ , written  $\mathcal{A} \subseteq \mathcal{B}$ , if  $A \subseteq B$ , and the interpretation of every function symbol and relation symbol in  $\tau$  is inherited from  $\mathcal{B}$ ; i.e., for each  $\alpha \in \tau$ ,  $\alpha^{\mathcal{A}}$  is the restriction  $\alpha^{\mathcal{B}} \upharpoonright A^k$  of  $\alpha^{\mathcal{B}}$  to  $A^k$ . We say that  $\mathcal{A}$  and  $\mathcal{B}$  are *isomorphic*, written  $\mathcal{A} \cong \mathcal{B}$ , if there exists a bijection (called an *isomorphism* between  $\mathcal{A}$  and  $\mathcal{B}$ )  $\pi : A \rightarrow B$  such that

- $\pi(f^{\mathcal{A}}(a_1, \dots, a_{\text{ar}(f)})) = f^{\mathcal{B}}(\pi(a_1), \dots, \pi(a_{\text{ar}(f)}))$ , for all function symbols  $f \in \tau$  and elements  $a_1, \dots, a_k \in A$ , and
- $(a_1, \dots, a_{\text{ar}(R)}) \in R^{\mathcal{A}} \iff (\pi(a_1), \dots, \pi(a_{\text{ar}(R)})) \in R^{\mathcal{B}}$ , for all relation symbols  $R \in \tau$  and elements  $a_1, \dots, a_k \in A$ .

We say that  $\mathcal{A}$  *embeds in*  $\mathcal{B}$ , written  $\mathcal{A} \preceq \mathcal{B}$ , if  $\mathcal{A}$  and some submodel of  $\mathcal{B}$  are isomorphic.

## B Graphoid axioms

We will use the following helping lemma in the proof of Theorem 12.

► **Lemma 22.** *Let  $R(\mathbf{X})$  be a  $K$ -relation, where  $K$  is  $\oplus$ -positive. Let  $t$  be a tuple of  $R$ , and let  $\mathbf{Y}, \mathbf{Z}$  be variable sets such that  $\mathbf{Z} \subseteq \mathbf{Y} \subseteq \mathbf{X}$ . Then,  $t[\mathbf{Y}] \in R[\mathbf{Y}]'$  implies  $t[\mathbf{Z}] \in R[\mathbf{Z}]'$ .*

**Proof.** By Lemma 4,  $R[\mathbf{Y}][\mathbf{Z}] = R[\mathbf{Z}]$ . Using Eq. (2) we have

$$R(t[\mathbf{Z}]) = \bigoplus_{\substack{u \in \text{Tuple}(\mathbf{Y}) \\ u[\mathbf{Z}] = t[\mathbf{Z}]}} R(u) = R(t[\mathbf{Y}]) \oplus \bigoplus_{\substack{u \in \text{Tuple}(\mathbf{Y}) \\ u[\mathbf{Z}] = t[\mathbf{Z}] \\ u[\mathbf{Y}] \neq t[\mathbf{Y}]}} R(u).$$

Since by assumption  $R(t[\mathbf{Y}]) \neq 0$ , we obtain by  $\oplus$ -positivity of  $K$  that  $R(t[\mathbf{Z}]) \neq 0$ , i.e.,  $t[\mathbf{Z}] \in R[\mathbf{Z}]'$ .  $\blacktriangleleft$

► **Theorem 12.** *Triviality, Symmetry, and Decomposition are sound for  $K$ -relations. Weak union and Contraction are sound for  $K$ -relations where  $K$  is positive and multiplicatively cancellative. Interaction is sound for total  $K$ -relations where  $K$  is positive and multiplicatively cancellative.*

**Proof.** Triviality and Symmetry are clearly sound for all  $K$ -teams. We thus consider only Decomposition, Weak union, and Contraction. Fix a  $K$ -relation  $R(\mathbf{V})$  over some variable set  $\mathbf{V}$  that contains  $\mathbf{XYZW}$ .

- Decomposition: Suppose  $R$  satisfies  $\mathbf{Y} \perp\!\!\!\perp \mathbf{ZW} \mid \mathbf{X}$ . Then, for all tuples  $t \in \text{Tuple}(\mathbf{V})$  it holds that

$$R(t[\mathbf{XY}])R(t[\mathbf{XZW}]) = R(t[\mathbf{X}])R(t[\mathbf{XYZW}]) \quad (10)$$

This implies

$$\begin{aligned} R(t[\mathbf{XY}])R(t[\mathbf{XZ}]) &= R(t[\mathbf{XY}]) \bigoplus_{\substack{t' \in \text{Tuple}(\mathbf{XZW}) \\ t'[\mathbf{XZ}] = t[\mathbf{XZ}]}} R(t'[\mathbf{XZW}]) \\ &= \bigoplus_{\substack{t' \in \text{Tuple}(\mathbf{XYZW}) \\ t'[\mathbf{XYZ}] = t[\mathbf{XYZ}]}} R(t'[\mathbf{XY}])R(t'[\mathbf{XZW}]) = \bigoplus_{\substack{t' \in \text{Tuple}(\mathbf{XYZW}) \\ t'[\mathbf{XYZ}] = t[\mathbf{XYZ}]}} R(t'[\mathbf{X}])R(t'[\mathbf{XYZW}]) \\ &= R(t[\mathbf{X}]) \bigoplus_{\substack{t' \in \text{Tuple}(\mathbf{XYZW}) \\ t'[\mathbf{XYZ}] = t[\mathbf{XYZ}]}} R(t'[\mathbf{XYZW}]) = R(t[\mathbf{X}])R(t[\mathbf{XYZ}]). \end{aligned}$$

Having showed

$$R(t[\mathbf{XY}])R(t[\mathbf{XZ}]) = R(t[\mathbf{X}])R(t[\mathbf{XYZ}]), \quad (11)$$

we conclude that  $R$  satisfies  $\mathbf{Y} \perp\!\!\!\perp \mathbf{Z} \mid \mathbf{X}$ .

- Weak union: Suppose again  $R$  satisfies  $\mathbf{Y} \perp\!\!\!\perp \mathbf{ZW} \mid \mathbf{X}$ , in which case Eq. (10) holds for all tuples  $t \in \text{Tuple}(\mathbf{V})$ . Multiplying both sides by  $R[t(\mathbf{XZ})]R[t(\mathbf{XYZ})]$  yields

$$\begin{aligned} &R(t[\mathbf{XY}])R(t[\mathbf{XZW}])R[t(\mathbf{XZ})]R[t(\mathbf{XYZ})] \\ &= R(t[\mathbf{X}])R(t[\mathbf{XYZW}])R[t(\mathbf{XZ})]R[t(\mathbf{XYZ})]. \end{aligned}$$

If  $R(t[\mathbf{XY}])R(t[\mathbf{XZ}]) \neq 0$ , we may apply Eq. (11), which is implied by Eq. (10), and multiplicative cancellativity to obtain

$$R(t[\mathbf{XZW}])R(t[\mathbf{XYZ}]) = R[t(\mathbf{XZ})]R[t(\mathbf{XYZW})]. \quad (12)$$

Suppose then  $R(t[\mathbf{XY}])R(t[\mathbf{XZ}]) = 0$ . Since  $K$  lacks zero divisors, either  $R(t[\mathbf{XY}]) = 0$  or  $R(t[\mathbf{XZ}]) = 0$ . By positivity and Lemma 22, it follows that  $R(t[\mathbf{XYZ}]) = 0$  and  $R(t[\mathbf{XYZW}]) = 0$ . In particular, both sides of Eq. (12) vanish. Hence we may conclude that  $R$  satisfies  $\mathbf{Y} \perp\!\!\!\perp \mathbf{W} \mid \mathbf{XZ}$ .

## 20:20 Conditional Independence on Semiring Relations

- Contraction: Suppose  $R$  satisfies  $Y \perp\!\!\!\perp Z \mid X$  and  $Y \perp\!\!\!\perp W \mid XZ$ , in which case we have Eq. (12) as well as

$$R(t[\mathbf{XY}])R(t[\mathbf{XZ}]) = R(t[\mathbf{X}])R(t[\mathbf{XYZ}]), \quad (13)$$

for all  $t \in \text{Dup}(\mathbf{V})$ . Multiplying both left-hand sides and right-hand sides of Eqs. (12) and (13) with one another yields

$$\begin{aligned} & R(t[\mathbf{XZW}])R(t[\mathbf{XYZ}])R(t[\mathbf{XY}])R(t[\mathbf{XZ}]) \\ &= R(t[\mathbf{XZ}])R(t[\mathbf{XYZW}])R(t[\mathbf{X}])R(t[\mathbf{XYZ}]). \end{aligned}$$

If  $R(t[\mathbf{XZ}])R(t[\mathbf{XYZ}]) \neq 0$ , we obtain Eq. (10) by multiplicative cancellativity. On the other hand, assuming  $R(t[\mathbf{XZ}])R(t[\mathbf{XYZ}]) = 0$  we have three cases:

1.  $R(t[\mathbf{XZ}]) = R(t[\mathbf{XYZ}]) = 0$ . Then  $R(t[\mathbf{XZW}]) = R(t[\mathbf{XYZW}]) = 0$  by positivity of  $R$  and Lemma 22, wherefore both sides of Eq. (10) vanish.
2.  $R(t[\mathbf{XZ}]) = 0$  and  $R(t[\mathbf{XYZ}]) \neq 0$ . Then  $R(t[\mathbf{X}]) = 0$  by positivity of  $K$  and Eq. (13). Again, both sides of Eq. (10) vanish.
3.  $R(t[\mathbf{XZ}]) \neq 0$  and  $R(t[\mathbf{XYZ}]) = 0$ . This time  $R(t[\mathbf{XY}]) = 0$  by positivity of  $K$  and Eq. (13), and once more we obtain Eq. (10).

Since Eq. (10) always holds, we conclude that  $R$  satisfies  $Y \perp\!\!\!\perp WZ \mid X$ .

- Interaction: Suppose  $R$  satisfies  $Y \perp\!\!\!\perp Z \mid XW$  and  $Y \perp\!\!\!\perp W \mid XZ$ . Then, we have

$$R(t[\mathbf{XZW}])R(t[\mathbf{XYW}]) = R(t[\mathbf{XW}])R(t[\mathbf{XYZW}]), \quad (14)$$

$$R(t[\mathbf{XYZ}])R(t[\mathbf{XZW}]) = R(t[\mathbf{XZ}])R(t[\mathbf{XYZW}]), \quad (15)$$

for all tuples  $t \in \text{Dup}(\mathbf{V})$ . By Proposition 1,  $K$  embeds in some semifield  $F$ . Given  $a \in K \setminus \{0\}$ , let us write  $a^{-1}$  for its multiplicative inverse in  $F$ . By positivity of  $K$ , totality of  $R$ , and Lemma 22, we observe  $R(t[\mathbf{U}]) \neq 0$  for all  $\mathbf{U} \subseteq \mathbf{V}$  and  $t \in \text{Dup}(\mathbf{V})$ . Thus, the expression  $R(t[\mathbf{U}' \mid t[\mathbf{U}]] := R(t[\mathbf{UU}'])R(t[\mathbf{U}])^{-1}$  is well defined for all  $\mathbf{U}, \mathbf{U}' \subseteq \mathbf{V}$  and  $t \in \text{Dup}(\mathbf{V})$ . We may now rewrite Eqs. (14) and (15) as

$$R(t[\mathbf{XYW} \mid t[\mathbf{XW}]] = R(t[\mathbf{XYZW} \mid t[\mathbf{XZW}]] = R(t[\mathbf{XYZ} \mid t[\mathbf{XZ}]]$$

from which we obtain

$$R(t[\mathbf{XYW}])R(t[\mathbf{XZ}]) = R(t[\mathbf{XYZ}])R(t[\mathbf{XW}])$$

for all  $t \in \text{Dup}(\mathbf{V})$ . Thus, for arbitrary  $t \in \text{Dup}(\mathbf{V})$ ,

$$\begin{aligned} R(t[\mathbf{XY}])R(t[\mathbf{XZ}]) &= R(t[\mathbf{XZ}]) \bigoplus_{\substack{t' \in \text{Dup}(\mathbf{XYW}) \\ t'[\mathbf{XY}] = t[\mathbf{XY}]} R(t'[\mathbf{XYW}]) \\ &= \bigoplus_{\substack{t' \in \text{Dup}(\mathbf{XYZW}) \\ t'[\mathbf{XYZ}] = t[\mathbf{XYZ}]} R(t'[\mathbf{XZ}])R(t'[\mathbf{XYW}]) \\ &= \bigoplus_{\substack{t' \in \text{Dup}(\mathbf{XYZW}) \\ t'[\mathbf{XYZ}] = t[\mathbf{XYZ}]} R(t'[\mathbf{XYZ}])R(t'[\mathbf{XW}]) \\ &= R(t[\mathbf{XYZ}]) \bigoplus_{\substack{t' \in \text{Dup}(\mathbf{XW}) \\ t'[\mathbf{X}] = t[\mathbf{X}]} R(t'[\mathbf{XW}]) \\ &= R(t[\mathbf{XYZ}])R(t[\mathbf{X}]), \end{aligned}$$

by which we conclude that  $R$  satisfies  $Y \perp\!\!\!\perp Z \mid X$ . Hence,  $R$  also satisfies  $Y \perp\!\!\!\perp ZW \mid X$  by soundness of Contraction.  $\blacktriangleleft$



# Subgraph Enumeration in Optimal I/O Complexity

Shiyuan Deng ✉

The Chinese University of Hong Kong, China

Yufei Tao ✉

The Chinese University of Hong Kong, China

---

## Abstract

Given a massive data graph  $G = (V, E)$  and a small pattern graph  $Q$ , the goal of *subgraph enumeration* is to list all the subgraphs of  $G$  isomorphic to  $Q$ . In the external memory (EM) model, it is well-known that every indivisible algorithm must perform  $\Omega(\frac{|E|^\rho}{M^{\rho-1}B})$  I/Os in the worst case, where  $M$  represents the number of words in (internal) memory,  $B$  denotes the number of words in a disk block, and  $\rho$  is the fractional edge covering number of  $Q$ . It has been a longstanding open problem to design an algorithm to match this lower bound. The state of the art is an algorithm in ICDT'23 that achieves an I/O complexity of  $O(\frac{|E|^\rho}{M^{\rho-1}B} \log_{M/B} \frac{|E|}{B})$  with high probability. In this paper, we remove the  $\log_{M/B} \frac{|E|}{B}$  factor, thereby settling the open problem when randomization is permitted.

**2012 ACM Subject Classification** Theory of computation → Graph algorithms analysis; Information systems → Join algorithms

**Keywords and phrases** Subgraph Enumeration, Conjunctive Queries, External Memory, Algorithms

**Digital Object Identifier** 10.4230/LIPIcs.ICDT.2024.21

**Funding** This work was supported in part by GRF projects 14207820, 14203421, and 14222822 from HKRGC.

## 1 Introduction

This paper revisits *subgraph enumeration*, a classical problem that has been extensively studied in computer science (see [2, 3, 5–8, 10, 13–19, 27, 30, 32, 40] and the references therein) and has many applications in database systems. The input is a simple undirected graph  $G = (V, E)$  called the *data graph* and another simple undirected graph  $Q = (V_Q, E_Q)$  called the *pattern*. The objective is to find all the pattern's *occurrences* in the data graph, or specifically, all the subgraphs<sup>1</sup> of  $G$  that are isomorphic to  $Q$ . We consider that (i)  $Q$  is connected (i.e., only a single connected component), and (ii)  $G$  has no isolated vertices (i.e., vertices with no incident edges), implying  $|V| \leq 2|E|$ .

We investigate the problem in the *external memory* (EM) model [1], the de facto model for studying I/O-efficient algorithms. A machine is equipped with  $M$  words of memory and an unbounded disk that has been partitioned into disjoint *blocks*, each comprising  $B$  words. The values of  $M$  and  $B$  satisfy  $M \geq 2B$ . An *I/O operation* either reads a disk block into memory or overwrites a disk block with  $B$  memory words. The *cost* of an algorithm is defined as the number of I/Os performed. CPU calculation is for free.

We require  $Q$  to have a constant size, namely,  $|V_Q| = O(1)$  (otherwise, even detecting whether  $G$  contains at least one occurrence of  $Q$  is NP-hard [9]). Conversely, the data graph  $G$  is presumed to have a gigantic size  $|E| > M$  and is provided under the adjacency format across  $O(|E|/B)$  blocks. An algorithm should report each occurrence  $G_{sub}$  of  $Q$  by “emission”. Specifically, the algorithm has the discretion to *emit*  $G_{sub}$  – for free – at any moment when all the edges of  $E_{sub}$  are memory resident, as long as  $G_{sub}$  is emitted only once throughout

---

<sup>1</sup> A *subgraph* of  $G$  is defined to be a graph  $G_{sub} = (V_{sub}, E_{sub})$  satisfying  $V_{sub} \subseteq V$  and  $E_{sub} \subseteq E$ .



the algorithm. Although not required to output the result to the disk, such an algorithm can be easily modified to do so in  $O(\text{OUT}/B)$  extra I/Os, where OUT is the total number of occurrences found.

Our discussion will focus on *indivisible* algorithms (also called *tuple-based* algorithms), which operate under the constraint that one I/O can bring  $O(B)$  edges into the memory, effectively prohibiting any compression tricks that attempt to pack  $\omega(B)$  edges into  $B$  words. Nearly all the existing subgraph enumeration algorithms are indivisible. Consequently, discerning the optimal I/O complexity attainable by this class provides valuable insight into the problem's characteristics.

**Previous Results.** An imperative parameter characterizing the computational hardness of subgraph enumeration is the *fractional edge covering number*  $\rho$  of the pattern  $Q = (V_Q, E_Q)$ . To understand this concept, imagine assigning, for each edge  $e = \{X, Y\} \in E_Q$ , a non-negative *weight*  $w_e$  such that, for each vertex  $X \in V_Q$ , the edges incident on  $X$  have a total weight at least 1, i.e.,  $\sum_{e \in E_Q: X \in e} w_e \geq 1$ . Then, the value of  $\rho$  is the smallest sum  $\sum_{e \in E_Q} w_e$  that can be achieved by all possible weight assignments. As a well-established lower bound [20, 23, 36], every (deterministic or randomized) indivisible algorithm needs  $\Omega(|E|^\rho / (M^{\rho-1}B))$  I/Os to solve the subgraph enumeration problem in the worst case.

Designing an algorithm to match this lower bound has been an intriguing endeavor. Next, we provide a chronicle of the milestones in the literature. In 2014, Pagh and Silvestri [36] studied the subgraph enumeration problem for  $Q = \text{triangle}$  (i.e., 3-clique) and presented two algorithms: the first is randomized and guarantees the I/O complexity  $O(|E|^{1.5} / (\sqrt{MB}))$  in expectation, while the second is deterministic and ensures an I/O cost of  $O(|E|^{1.5} / (\sqrt{MB}) \cdot \log_{M/B} \frac{|E|}{B})$ . In 2015, Hu, Qiao, and Tao [22] (long version in [23]) managed to improve the deterministic bound to  $O(|E|^{1.5} / (\sqrt{MB}))$ . Because  $\rho = 1.5$  for  $Q = \text{triangle}$ , the algorithm of [22] is asymptotically optimal (and so is the randomized algorithm of [36] in expected performance).

In 2016, Hu and Yi [20] considered the scenario where the pattern  $Q$  is an (arbitrary) tree and gave a deterministic algorithm with I/O complexity  $O(\frac{|E|^\rho}{M^{\rho-1}B} \log_{M/B} \frac{|E|}{B})$ . In 2017, Ketsman and Suciú [25] obtained an algorithm that, given an arbitrary pattern  $Q$ , solves the subgraph enumeration problem in  $O(\frac{|E|^\rho}{M^{\rho-1}B} \text{polylog } |E|)$  I/Os with high probability (i.e., with probability at least  $1 - 1/|E|^c$ , where  $c$  can be an arbitrarily large constant decided before running the algorithm), provided that  $M \geq |E|^{\Omega(1)}$ . In 2020, Tao [38] found a simpler algorithm attaining the same I/O complexity as [25]. Another main contribution of [38] is its establishment of the *isolated cartesian product theorem* (the ICP-theorem) (see also [26]), which serves as the main weapon in analyzing the performance of a method called *heavy-light decomposition*. This method underlies the algorithm of [38] and all the subsequent works (including ours) on the topic.

In 2023, Deng, Silvestri, and Tao [12] presented new progress that came close to achieving the optimal I/O bound for a general pattern  $Q = (V_Q, E_Q)$ . Their algorithm finishes in  $O(\frac{|E|^\rho}{M^{\rho-1}B} + \frac{|E|^{k/2}}{M^{k/2-1}B} \log_{M/B} \frac{|E|}{B})$  I/Os with high probability, where the parameter  $k$  represents the size of  $V_Q$ . In general, we know  $\rho \geq k/2$  [37]. For a pattern  $Q$  where  $\rho$  is strictly greater than  $k/2$ , the I/O cost of [12] can be simplified to the optimal bound  $O(|E|^\rho / (M^{\rho-1}B))$ . However, when  $\rho = k/2$ , the I/O bound becomes  $O(\frac{|E|^\rho}{M^{\rho-1}B} \log_{M/B} \frac{|E|}{B})$ , which is away from optimality by a  $\log_{M/B} \frac{|E|}{B}$  factor. It is worth mentioning that the relationship  $\rho = k/2$  is satisfied by many patterns  $Q$ , important examples of which include  $k$ -cycles and  $k$ -cliques.

Unlike in the EM model, subgraph enumeration has been well understood in the traditional RAM model, where the problem can be settled in  $O(|E|^\rho)$  time [34, 35, 39], which is known to be worst-case optimal; see also [4, 11, 24, 28, 29, 31, 33] for algorithms with  $O(|E|^\rho \text{polylog } |E|)$

■ **Table 1** A summary of the previous results and ours.

pattern $Q$	I/O cost in big- $O$	source	remark
triangle	$ E ^{1.5}/(\sqrt{MB})$ expected	[36]	
triangle	$\frac{ E ^{1.5}}{\sqrt{MB}} \log_{M/B} \frac{ E }{B}$	[36]	
triangle	$ E ^{1.5}/(\sqrt{MB})$	[21, 23]	
acyclic	$\frac{ E ^\rho}{M^{\rho-1}B} \log_{M/B} \frac{ E }{B}$	[20]	
arbitrary	$\frac{ E ^\rho}{M^{\rho-1}B} \cdot \text{polylog }  E $ w.h.p.	[25, 26, 38]	needs $M \geq  E ^{\Omega(1)}$
arbitrary	$\frac{ E ^\rho}{M^{\rho-1}B} \log_{M/B} \frac{ E }{B}$ w.h.p.	[12]	
<b>arbitrary</b>	<b><math> E ^\rho/(M^{\rho-1}B)</math> w.h.p.</b>	<b>ours</b>	<b>optimal</b>

time. In general, it is non-trivial to translate a RAM algorithm into an efficient external memory counterpart: a direct translation of the solutions in [4, 11, 24, 28, 29, 31, 33–35, 39] will result in a huge I/O complexity of  $\Omega(|E|^\rho)$ . Indeed, the aforementioned subgraph enumeration algorithms in EM harbor numerous ideas that are purposed for accessing data in *blocks* and are thus not required in RAM.

**Our Results.** This paper’s main contribution is an algorithm that solves the subgraph enumeration problem in  $O(|E|^\rho/(M^{\rho-1}B))$  I/Os with high probability (and, hence, also in expectation). Although this improves the result of [12] by “only” a logarithmic factor, our algorithm is the first whose I/O complexity matches the lower bound  $\Omega(|E|^\rho/(M^{\rho-1}B))$  for any pattern  $Q$ . See Table 1 for a comparison between our result and the existing ones.

**Math Conventions.** For an integer  $x \geq 1$ , the notation  $[x]$  represents the set  $\{1, 2, \dots, x\}$ . We define  $\text{sort}(n)$  to be the I/O complexity of sorting  $n$  elements; it is known [1] that  $\text{sort}(n) = O(1 + \lceil \frac{n}{B} \rceil \log_{M/B} \lceil \frac{n}{B} \rceil)$ . We use double curly braces to represent multi-sets, e.g.,  $\{\{1, 1, 1, 2, 2, 3\}\}$  is a multi-set with 6 elements. Symbol  $\mathbb{N}$  represents the set of integers.

## 2 The Heavy-Light Algorithmic Framework

This section will introduce the *heavy-light decomposition* technique, a framework that transforms subgraph enumeration into a join on binary relations. To lay the groundwork for a formal discussion, we will begin by reviewing the relevant concepts of joins and their connections to subgraph enumeration in Section 2.1. Then, Section 2.2 will delve into the specifics of the decomposition technique at a level sufficient for our technical development.

### 2.1 Reducing Subgraph Enumeration to Joins

Let  $\mathbf{att}$  represent an arbitrary infinite set where each element is called an *attribute*. Given any set  $U \subseteq \mathbf{att}$ , we define a *tuple* over  $U$  as a function  $\mathbf{t} : U \rightarrow \mathbb{N}$ . Given any subset  $U_{sub}$  of  $U$ , we designate  $\mathbf{t}[U_{sub}]$  as the tuple  $\mathbf{t}_{sub}$  over  $U_{sub}$  such that  $\mathbf{t}_{sub}(X) = \mathbf{t}(X)$  holds true for every  $X \in U_{sub}$ .

A relation is a set  $R$  comprising tuples over the same set  $U$  of attributes. The set  $U$  is called the *schema* of  $R$ , a fact we denote as  $U = \text{schema}(R)$ . We say that  $R$  is *unary* if  $|\text{schema}(R)| = 1$ , or *binary* if  $|\text{schema}(R)| = 2$ . Given an integer value  $x$  and an attribute  $X \in \text{schema}(R)$ , the *degree of  $x$  under  $X$  in  $R$*  is the number of tuples  $\mathbf{t} \in R$  satisfying

## 21:4 Subgraph Enumeration in Optimal I/O Complexity

$t(X) = x$ . We say that the attribute  $X$  has *degree* at most  $\lambda$  in  $R$  if all values have degrees bounded by  $\lambda$  under  $X$  in  $R$ . For each attribute  $X \in \text{schema}(R)$ , define its *active domain under  $R$*  – represented as  $\mathbf{adom}_R(X)$  – as  $\Pi_X(R)$ , where  $\Pi$  is the standard projection operator in relational algebra.

A *join* is formalized as a set  $\mathcal{Q}$  of relations. If we denote  $\text{schema}(\mathcal{Q}) = \bigcup_{R \in \mathcal{Q}} \text{schema}(R)$ , the join result – denoted as  $\text{join}(\mathcal{Q})$  – can be defined as a relation over  $\text{schema}(\mathcal{Q})$ :

$$\text{join}(\mathcal{Q}) = \{\text{tuple } \mathbf{t} \text{ over } \text{schema}(\mathcal{Q}) \mid \forall R \in \mathcal{Q} : \mathbf{t}[\text{schema}(R)] \in R\}.$$

The *input size*  $N$  of  $\mathcal{Q}$  is the total number of tuples in all the relations of  $\mathcal{Q}$ , namely,  $N = \sum_{R \in \mathcal{Q}} |R|$ . The join  $\mathcal{Q}$  is *schema-clean* if it has no two relations such that the schema of one relation contains that of the other. For each attribute  $X \in \text{schema}(\mathcal{Q})$ , define its *active domain under  $\mathcal{Q}$*  – represented as  $\mathbf{adom}_{\mathcal{Q}}(X)$  – as  $\bigcup_{R \in \mathcal{Q} : X \in \text{schema}(R)} \mathbf{adom}_R(X)$ , namely, the set of values that appear under  $X$  in at least one relation of  $\mathcal{Q}$ .

Every join  $\mathcal{Q}$  defines a *schema graph*, which is a hypergraph  $\mathcal{G} = (\mathcal{X}, \mathcal{E})$  where  $\mathcal{X} = \text{schema}(\mathcal{Q})$  and  $\mathcal{E} = \{\{\text{schema}(R) \mid R \in \mathcal{Q}\}\}$ . Throughout the paper, we will refer to the elements in  $\mathcal{X}$  as “attributes” and the elements in  $\mathcal{E}$  as “hyperedges”. Note that  $\mathcal{E}$  is a multi-set: even if two relations  $R_1, R_2 \in \mathcal{Q}$  share a common schema, we still treat  $\text{schema}(R_1)$  and  $\text{schema}(R_2)$  as different hyperedges in  $\mathcal{E}$ . As another noteworthy remark, if a relation  $R \in \mathcal{Q}$  is unary, the hyperedge  $\text{schema}(R) \in \mathcal{E}$  has only one attribute. Finally, we say that two attributes  $X_1, X_2 \in \mathcal{X}$  are *adjacent* in  $\mathcal{G}$  if they co-appear in at least one hyperedge in  $\mathcal{E}$ .

In EM, the *join enumeration* problem is formalized as follows. Let  $\mathcal{Q}$  be a schema-clean join whose schema graph  $\mathcal{G} = (\mathcal{X}, \mathcal{E})$  has  $O(1)$  attributes. Initially, the tuples of each relation  $R \in \mathcal{Q}$  – which we call the *raw tuples* – are provided in  $O(\lceil |R|/B \rceil)$  consecutive blocks on the disk, and  $\mathcal{G}$  is provided in memory using  $O(1)$  words. Similar to subgraph enumeration, an algorithm is not required to write the result to the disk. Instead, it suffices to do the reporting through emission. Specifically, for each tuple  $\mathbf{t} \in \text{join}(\mathcal{Q})$ , the algorithm can *emit*  $\mathbf{t}$  for free at any moment when the raw tuple  $\mathbf{t}[\text{schema}(R)]$  exists in memory for every  $R \in \mathcal{Q}$ , but  $\mathbf{t}$  should be emitted only once throughout the algorithm. A randomized algorithm is said to guarantee an I/O bound “with high probability” if the bound holds asymptotically with a probability at least  $1 - 1/N^c$ , where  $c$  can be any arbitrarily large constant decided before running the algorithm. We consider only *indivisible* algorithms where each I/O can bring only  $O(B)$  raw tuples into memory.

The following lemma was proved in [12]:

► **Lemma 1** ([12]). *Consider any input to subgraph enumeration with data graph  $G = (V, E)$  and pattern graph  $Q$ . It is possible to construct a join  $\mathcal{Q}$  with schema graph  $Q$  and input size  $N = \Theta(|E|)$  such that, if we can emit all the result tuples of  $\mathcal{Q}$  using an indivisible algorithm in  $T_{\text{join}}$  I/Os with high probability, then we can design an indivisible algorithm to solve the original subgraph enumeration problem using  $T_{\text{join}} + O(\lceil |E|/B \rceil)$  I/Os with high probability.*

As implied by the lemma’s statement, the join  $\mathcal{Q}$  constructed contains only binary relations, all of which have distinct schemas. Henceforth, we will concentrate on devising an algorithm to process any schema-clean join  $\mathcal{Q}$  on binary relations in  $T_{\text{join}} = O(N^\rho / (M^{\rho-1} B))$  I/Os with high probability, where  $\rho$  is the fractional edge-covering number of the schema graph of  $\mathcal{Q}$ . Once this is done, we will have obtained a subgraph enumeration algorithm that achieves the I/O complexity  $O(|E|^\rho / (M^{\rho-1} B))$  with high probability. In the rest of the paper, we consider that  $\mathcal{Q}$  has at least two relations (if  $\mathcal{Q}$  has only one relation, the join requires no I/Os because the relation itself is the join result).

## 2.2 Heavy-Light Decomposition

Given a join  $\mathcal{Q}$ , the *heavy-light decomposition method* conceptually partitions the tuples  $\mathbf{t}$  in the join result  $join(\mathcal{Q})$  by (i) the number of “high-degree” values used in  $\mathbf{t}$  and (ii) the specific attributes under which those values appear. The method then concentrates on computing the result tuples in each partition separately. Presented below is a version of the method that was introduced in [38] and deployed in the subsequent works [12, 26].

Suppose that we are given a join  $\mathcal{Q}$  where there are at least two relations, all the relations are binary, and their schemas are distinct. Denote by  $\mathcal{G} = (\mathcal{X}, \mathcal{E})$  the schema graph of  $\mathcal{Q}$ . For each hyperedge  $e \in \mathcal{E}$ , we use  $R_e$  to represent the (only) relation in  $\mathcal{Q}$  whose schema is  $e$ .

Define

$$\lambda = \sqrt{NM} \quad (1)$$

where  $N$  is the input size of  $\mathcal{Q}$  and  $M$  is the memory size. An integer  $x$  is *heavy* if the degree of  $x$  under an attribute of any relation is no less than  $\lambda$ ; otherwise, the integer is *light*. The total number of heavy integer values is bounded by  $O(N/\lambda)$ .

**Configurations.** Fix an arbitrary subset  $\mathcal{H} \subseteq \mathcal{X}$ . We call each attribute of  $\mathcal{H}$  a *heavy* attribute. A *configuration* of  $\mathcal{H}$  is a tuple  $\boldsymbol{\eta}$  over  $\mathcal{H}$  such that  $\boldsymbol{\eta}(X)$  is a heavy value for every  $X \in \mathcal{H}$ . We denote by  $config(\mathcal{H})$  the set of “active configurations”  $\boldsymbol{\eta}$  satisfying  $\boldsymbol{\eta}[e] \in R_e$  for every hyperedge  $e \in \mathcal{E}$  subsumed by  $\mathcal{H}$  (i.e.,  $e \subseteq \mathcal{H}$ ); note that if no such hyperedges exist, then every configuration of  $\mathcal{H}$  is active. It is easy to see that  $|config(\mathcal{H})| = O((N/\lambda)^{|\mathcal{H}|}) = O((N/M)^{|\mathcal{H}|/2})$ , plugging in the value of  $\lambda$  in (1).

**Residual Joins.** Fix any subset  $\mathcal{H} \subseteq \mathcal{X}$  and any active configuration  $\boldsymbol{\eta} \in config(\mathcal{H})$ . Next, we will formulate a “residual join” to compute the tuples  $\mathbf{u} \in join(\mathcal{Q})$  that “agree” with  $\boldsymbol{\eta}$  on every heavy attribute. We say that a hyperedge  $e \in \mathcal{E}$  is *relevant* to  $\mathcal{H}$  if  $e \setminus \mathcal{H} \neq \emptyset$ , namely,  $e$  is not subsumed by  $\mathcal{H}$ . For every relevant  $e \in \mathcal{E}$ , define  $R_e(\boldsymbol{\eta})$  as the relation that comprises every tuple  $\mathbf{t} \in R_e$  satisfying

- $\mathbf{t}(X) = \boldsymbol{\eta}(X)$  for all  $X \in e \cap \mathcal{H}$ ;
- $\mathbf{t}(X)$  is light for every  $X \in e \setminus \mathcal{H}$ .

Namely,  $\mathbf{t}(X)$  must take *the* heavy value  $\boldsymbol{\eta}(X)$  if  $X$  is heavy; otherwise,  $\mathbf{t}(X)$  must be light.

Once  $R_e(\boldsymbol{\eta})$  is ready, we can define the *residual relation of  $e$  under  $\boldsymbol{\eta}$*  – denoted as  $R'_e(\boldsymbol{\eta})$  – via a simple projection<sup>2</sup>:

$$R'_e(\boldsymbol{\eta}) = \Pi_{e \setminus \mathcal{H}}(R_e(\boldsymbol{\eta})). \quad (2)$$

Now, by putting together the residual relations defined by all the relevant hyperedges  $e$ , we obtain the *residual join of  $\boldsymbol{\eta}$*  – denoted as  $\mathcal{Q}'(\boldsymbol{\eta})$  – which can be formalized as

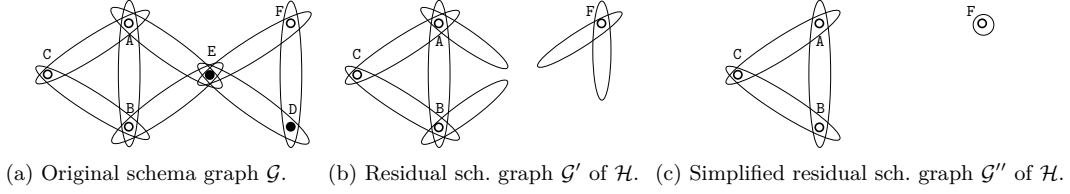
$$\mathcal{Q}'(\boldsymbol{\eta}) = \{R'_e(\boldsymbol{\eta}) \mid e \in \mathcal{E}, e \text{ relevant to } \mathcal{H}\}. \quad (3)$$

We will use  $N_{\boldsymbol{\eta}}$  to represent the input size of  $\mathcal{Q}'(\boldsymbol{\eta})$ , that is

$$N_{\boldsymbol{\eta}} = \sum_{R \in \mathcal{Q}'(\boldsymbol{\eta})} |R|. \quad (4)$$

It is important to note that the relations in  $\mathcal{Q}'(\boldsymbol{\eta})$  consist solely of light values.

<sup>2</sup> Strictly speaking, a tuple in  $R'_e(\boldsymbol{\eta})$  may no longer be a raw tuple (recall that raw tuples are those in the input relations of  $\mathcal{Q}$ ). This would create an issue when the need arises to verify our algorithms’ indivisibility. The issue, however, can be easily fixed by “augmenting” each tuple  $\mathbf{u} \in R'_e(\boldsymbol{\eta})$  with  $\boldsymbol{\eta}$ , thereby recording the raw tuple of  $R_e$  corresponding to  $\mathbf{u}$ . In the subsequent discussion, we will no longer be concerned with such (pedantic) details, except to mention that all our join algorithms can be implemented in an indivisible manner without affecting the claimed I/O complexities.



■ **Figure 1** Illustration of the heavy-light decomposition method ( $\mathcal{H} = \{D, E\}$ ).

To “visualize” the schemas of the relations in  $\mathcal{Q}'(\eta)$ , define  $\mathcal{G}' = (\mathcal{X}', \mathcal{E}')$  as the hypergraph obtained by discarding the heavy attributes from  $\mathcal{G} = (\mathcal{X}, \mathcal{E})$ , namely,  $\mathcal{X}' = \mathcal{X} \setminus \mathcal{H}$  and  $\mathcal{E}' = \{e \setminus \mathcal{H} \mid e \in \mathcal{E} \text{ and } e \setminus \mathcal{H} \neq \emptyset\}$  (note:  $\mathcal{E}'$  is a multi-set). This  $\mathcal{G}'$  – called the *residual schema graph* of  $\mathcal{H}$  – is the schema graph of  $\mathcal{Q}'(\eta)$ ; this is true regardless of  $\eta$ .

► **Example 2.** For an illustration, Figure 1a presents the schema graph  $\mathcal{G} = (\mathcal{X}, \mathcal{E})$  of a join  $\mathcal{Q}$ , where  $\mathcal{X} = \{A, B, C, D, E, F\}$  and  $\mathcal{E} = \{AB, AC, AE, BC, BE, DE, DF, EF\}$ . Set  $\mathcal{H} = \{D, E\}$ ; the two heavy attributes  $D$  and  $E$  are colored black in the figure. Let  $\eta$  be an arbitrary active configuration of  $\mathcal{H}$ . All the hyperedges are relevant to  $\mathcal{H}$ , except  $DE$ . The residual relation of  $AB$  under  $\eta$  – that is,  $R'_{AB}(\eta)$  – is the same as  $R_{AB}(\eta)$ , which in turn is simply the relation  $R_{AB}$  in  $\mathcal{Q}$ . The residual relation of  $AE$  under  $\eta$  – that is,  $R'_{AE}(\eta)$  – has only one attribute  $A$  and contains every light value  $\mathbf{a}$  such that  $R_{AE}$  has a tuple  $\mathbf{u}$  with  $\mathbf{u}(A) = \mathbf{a}$  and  $\mathbf{u}(E) = \eta(E)$ . Figure 1b shows the residual schema graph  $\mathcal{G}'$ , which is the schema graph of the residual join  $\mathcal{Q}'(\eta)$  of  $\eta$ .

By considering all subsets  $\mathcal{H} \subseteq \mathcal{X}$ , we have

$$\text{join}(\mathcal{Q}) = \bigcup_{\mathcal{H}} \left( \bigcup_{\eta \in \text{config}(\mathcal{H})} \text{join}(\mathcal{Q}'(\eta)) \times \{\eta\} \right). \quad (5)$$

This transforms the evaluation of the original join  $\mathcal{Q}$  into computing the residual joins.

**Simplified Residual Joins.** Again, fix any subset  $\mathcal{H} \subseteq \mathcal{X}$  and one arbitrary configuration  $\eta \in \text{config}(\mathcal{H})$ . One issue arising from computing the residual join  $\mathcal{Q}'(\eta)$  is that the join may not be schema-clean, i.e.,  $\mathcal{Q}'(\eta)$  may contain two relations where one relation’s schema encompasses that of the other. The second phase of the heavy-light decomposition framework is to convert each  $\mathcal{Q}'(\eta)$  to its schema-clean version. Next, we explain how this is done.

Define  $\mathcal{L} = \mathcal{X} \setminus \mathcal{H}$ ; we will call each attribute in  $\mathcal{L}$  a *light* attribute (as opposed to the heavy attributes in  $\mathcal{H}$ ). We say that a light attribute  $X \in \mathcal{L}$  is a *border attribute* if it is adjacent in  $\mathcal{G}$  to at least one heavy attribute in  $\mathcal{H}$ . For each such attribute  $X$ , define

$$R''_X(\eta) = \bigcap_{e \in \mathcal{E}: X \in e} \Pi_X(R'_e(\eta)). \quad (6)$$

To understand the intuition behind, here we look at all the edges (of  $\mathcal{G}$ ) containing  $X$  and examine their residual relations under  $\eta$ . The relation  $R''_X(\eta)$  collects every integer that appears under attribute  $X$  in all those residual relations.

Now consider each hyperedge  $e = \{X_1, X_2\}$  in  $\mathcal{G}$  where both  $X_1$  and  $X_2$  are light attributes. Note that  $e$  is also a hyperedge in  $\mathcal{E}'$ . Construct a relation  $R''_e(\eta) \subseteq R'_e(\eta)$  as follows:

- If  $X_1$  and  $X_2$  are both border attributes, then  $R''_e(\eta) = R'_e(\eta) \bowtie R''_{X_1}(\eta) \bowtie R''_{X_2}(\eta)$ , where  $\bowtie$  is the standard natural join operator in relational algebra;
- If only  $X_1$  is a border attribute, then  $R''_e(\eta) = R'_e(\eta) \bowtie R''_{X_1}(\eta)$ ;
- If only  $X_2$  is a border attribute, then  $R''_e(\eta) = R'_e(\eta) \bowtie R''_{X_2}(\eta)$ ;
- If neither is a border attribute, then  $R''_e(\eta) = R'_e(\eta)$ .

It is safe to discard the tuples in  $R'_e(\eta) \setminus R''_e(\eta)$  because they cannot contribute to the result of the residual join  $\mathcal{Q}'(\eta)$ . We call  $R''_e(\eta)$  the *simplified residual relation of  $e$  under  $\eta$* .

A border attribute  $X \in \mathcal{X}$  may become *isolated* in  $\mathcal{G}'$ , meaning that  $X$  is not adjacent to any other vertex in  $\mathcal{G}'$  (i.e.,  $X$  appears only in unary hyperedges of  $\mathcal{E}'$ ). This happens when  $X$  is not adjacent to any light attribute in  $\mathcal{G}$  (i.e.,  $X$  is adjacent to only heavy attributes in  $\mathcal{G}$ ). Let  $\mathcal{I}$  be the set of isolated attributes in  $\mathcal{G}'$ .

The *simplified residual join* of  $\eta$  – denoted as  $\mathcal{Q}''(\eta)$  – can now be formalized as

$$\mathcal{Q}''(\eta) = \{R''_e(\eta) \mid \text{binary } e \in \mathcal{E}'\} \cup \{R''_X(\eta) \mid X \in \mathcal{I}\}. \quad (7)$$

To visualize the schemas of the relations in  $\mathcal{Q}''(\eta)$ , define  $\mathcal{G}'' = (\mathcal{X}'', \mathcal{E}'')$  where  $\mathcal{X}'' = \mathcal{X}' = \mathcal{L}$  and  $\mathcal{E}''$  contains (i) all the binary edges of  $\mathcal{E}'$  and (ii) one unary edge for each isolated attribute. This  $\mathcal{G}''$  is the schema graph of  $\mathcal{Q}''(\eta)$ , regardless of  $\eta \in \text{config}(\mathcal{H})$ .

► **Example 3.** *Let us continue the discussion in Example 2. In Figure 1a,  $\mathcal{L} = \{A, B, C, F\}$ , and among these attributes,  $A, B$  and  $F$  are border attributes, but only  $F$  is an isolated attribute. The relation  $R''_A$  is the intersection of  $\Pi_A(R'_{AB}(\eta)) = \Pi_A(R_{AB})$ ,  $\Pi_A(R'_{AC}(\eta)) = \Pi_A(R_{AC})$ , and  $R'_{AE}(\eta)$  (recall that  $R'_{AE}(\eta)$  contains only one attribute, i.e.,  $A$ ). The simplified residual relation of  $AB$  – that is,  $R''_{AB}(\eta)$  – comprises every tuple  $\mathbf{u} \in R'_{AB}(\eta)$  such that  $\mathbf{u}(A) \in R''_A$  and  $\mathbf{u}(B) \in R''_B$ . The simplified residual relation of  $BC$  – that is,  $R''_{BC}(\eta)$  – comprises every tuple  $\mathbf{u} \in R'_{BC}(\eta)$  such that  $\mathbf{u}(B) \in R''_B$ . Figure 1c shows the simplified residual schema graph  $\mathcal{G}''$ , which is the schema graph of the simplified residual join  $\mathcal{Q}''(\eta)$  of  $\eta$ .*

It is easy to verify that  $\text{join}(\mathcal{Q}''(\eta)) = \text{join}(\mathcal{Q}'(\eta))$ . Combining this fact with (5), we have

$$\text{join}(\mathcal{Q}) = \bigcup_{\mathcal{H}} \left( \bigcup_{\eta \in \text{config}(\mathcal{H})} \text{join}(\mathcal{Q}''(\eta)) \times \{\eta\} \right). \quad (8)$$

which transforms the evaluation of the original join  $\mathcal{Q}$  into computing  $\text{join}(\mathcal{Q}''(\eta)) \times \{\eta\}$  for all  $\eta \in \text{config}(\mathcal{H})$  and  $\mathcal{H} \subseteq \mathcal{X}$ .

A main contribution of [12] is the following lemma.

► **Lemma 4** ([12]). *Let  $\mathcal{Q}$  be a schema-clean join on at least two binary relations whose schema graph is  $\mathcal{G} = (\mathcal{X}, \mathcal{E})$ . Let  $k = |\mathcal{X}|$ ,  $N$  be the input size of  $\mathcal{Q}$ , and  $\rho$  be the fractional edge covering number of  $\mathcal{G}$ . Suppose that, for any  $\mathcal{H} \subseteq \mathcal{X}$ , using  $T_{\mathcal{H}}$  I/Os in total we can produce*

- $\text{config}(\mathcal{H})$  in consecutive disk blocks;
- each input relation of  $\mathcal{Q}''(\eta)$  in consecutive disk blocks for every active configuration  $\eta \in \text{config}(\mathcal{H})$ .

*Then, there is an algorithm for computing  $\text{join}(\mathcal{Q})$  using  $\sum_{\mathcal{H} \subseteq \mathcal{X}} T_{\mathcal{H}} + O(N^{\rho}/(M^{\rho-1}B))$  I/Os.*

The algorithm of [12] necessitates  $\sum_{\mathcal{H} \subseteq \mathcal{X}} T_{\mathcal{H}} = O(\frac{N^{k/2}}{M^{k/2-1}B} \log_{M/B} \frac{N}{B})$  I/Os, as is the culprit behind the algorithm's sub-optimality. Our mission in this work is:

$$\text{reduce } \sum_{\mathcal{H} \subseteq \mathcal{X}} T_{\mathcal{H}} \text{ to } O(N^{k/2}/(M^{k/2-1}B)). \quad (9)$$

### 3 Binary-Relation Joins in a Small Domain

In this and the next sections, we will discuss two standalone problems that are intriguing in their own right. The solutions to those problems will play a crucial role in our approach to achieving the objective given in (9).



This section will study join enumeration in the special scenario where the number of distinct values is limited. The rest of the section serves as a proof of:

► **Lemma 5.** *Let  $\mathcal{Q}$  be a schema-clean join on binary relations, and let  $\mathcal{G} = (\mathcal{X}, \mathcal{E})$  be its schema graph. If  $|\mathbf{adom}_{\mathcal{Q}}(X)| \leq D$  for every attribute  $X \in \mathcal{X}$ , we can emit all the result tuples of the join  $\mathcal{Q}$  in  $O((1 + \frac{D}{\sqrt{M}})^k \cdot \frac{M}{B} + \text{sort}(N))$  I/Os, where  $k = |\mathcal{X}|$ ,  $N$  is the input size of  $\mathcal{Q}$ ,  $M$  is the memory size, and  $B$  is the disk block size.*

Recall from Section 2 that  $\mathbf{adom}_{\mathcal{Q}}(X)$  is the active domain of  $X$  under  $\mathcal{Q}$ . In general, the value of  $D$  (i.e., the maximum active domain size) can reach  $\Omega(N)$ , in which case Lemma 5 is not particularly useful: it is worse than the  $O(N^\rho/(M^{\rho-1}B))$  I/O bound that will be ultimately established in this paper. However, the lemma's strength is reflected in the scenario where  $D$  is small. For example, for  $D = \sqrt{N}$  and  $k \geq 3$ , the I/O complexity of Lemma 5 becomes  $O(N^{k/2}/(M^{k/2-1}B))$ , which is never worse, but can be considerably better, than  $O(N^\rho/(M^{\rho-1}B))$  because the value of  $\rho$  falls in the range from  $k/2$  to  $k-1$  [37].

We now proceed to prove the lemma. For each attribute  $X \in \mathcal{X}$ , we obtain and sort the active domain  $\mathbf{adom}_{\mathcal{Q}}(X)$  in  $O(\text{sort}(N))$  I/Os. Then, by scanning (the sorted)  $\mathbf{adom}_{\mathcal{Q}}(X)$ , we partition the integer domain  $\mathbb{N}$  into a set  $S_X^{\text{intv}}$  of disjoint intervals such that each interval covers  $\lfloor \sqrt{M}/c \rfloor$  values of  $\mathbf{adom}_{\mathcal{Q}}(X)$  – where  $c$  is a constant to be chosen later – except possibly one interval that may cover less values. The size of  $S_X^{\text{intv}}$  is  $O(1 + D/\sqrt{M})$  because  $|\mathbf{adom}_{\mathcal{Q}}(X)| \leq D$ .

Let  $X_1, X_2, \dots, X_k$  be an arbitrary ordering of the  $k$  attributes in  $\mathcal{X}$ . Consider any hyperedge  $e = \{X_i, X_j\}$  of  $\mathcal{E}$  ( $1 \leq i < j \leq k$ ). Given an interval  $I_i \in S_{X_i}^{\text{intv}}$  and an interval  $I_j \in S_{X_j}^{\text{intv}}$ , define

$$R_e(I_i, I_j) = \{\mathbf{u} \in R_e \mid \mathbf{u}(X_i) \in I_i \text{ and } \mathbf{u}(X_j) \in I_j\}.$$

Recall that  $R_e$  is the (only) relation in  $\mathcal{Q}$  with schema  $e$ . As  $I_i$  (resp.,  $I_j$ ) covers  $\lfloor \sqrt{M}/c \rfloor$  values of  $\mathbf{adom}_{\mathcal{Q}}(X_i)$  (resp.,  $\mathbf{adom}_{\mathcal{Q}}(X_j)$ ), the set  $R_e(I_i, I_j)$  can contain at most  $(\lfloor \sqrt{M}/c \rfloor)^2 \leq M/c^2$  tuples. For each interval combination  $(I_1, I_2, \dots, I_k) \in S_{X_1}^{\text{intv}} \times S_{X_2}^{\text{intv}} \times \dots \times S_{X_k}^{\text{intv}}$ , define a sub-join

$$\mathcal{Q}(I_1, I_2, \dots, I_k) = \{R_e(I_i, I_j) \mid e = \{X_i, X_j\} \in \mathcal{E}\}.$$

It is easy to verify that

$$\text{join}(\mathcal{Q}) = \bigcup_{(I_1, I_2, \dots, I_k) \in S_{X_1}^{\text{intv}} \times S_{X_2}^{\text{intv}} \times \dots \times S_{X_k}^{\text{intv}}} \text{join}(\mathcal{Q}(I_1, I_2, \dots, I_k)). \quad (10)$$

Let us assume, for now, that, given a hyperedge  $e = \{X_i, X_j\} \in \mathcal{E}$ , any interval  $I_i \in S_{X_i}^{\text{intv}}$ , and any interval  $I_j \in S_{X_j}^{\text{intv}}$ , we can load  $R_e(I_i, I_j)$  from the disk into memory using  $O(M/B)$  I/Os. Fix an arbitrary interval combination  $(I_1, I_2, \dots, I_k) \in S_{X_1}^{\text{intv}} \times S_{X_2}^{\text{intv}} \times \dots \times S_{X_k}^{\text{intv}}$ . If we add up the size of  $R_e(I_i, I_j)$  for all hyperedges  $e = \{X_i, X_j\} \in \mathcal{E}$ , the total size is at most  $|\mathcal{E}| \cdot M/c^2$ , which is at most  $M$  by setting the constant  $c$  sufficiently large. Hence, in  $O(M/B) \cdot |\mathcal{E}| = O(M/B)$  I/Os, we can load into memory the set  $R_e(I_i, I_j)$  of every hyperedge  $e = \{X_i, X_j\} \in \mathcal{E}$ . This permits us to emit the result tuples of  $\mathcal{Q}(I_1, I_2, \dots, I_k)$  with no more I/Os. As the cartesian product  $S_{X_1}^{\text{intv}} \times S_{X_2}^{\text{intv}} \times \dots \times S_{X_k}^{\text{intv}}$  has a size of  $O((1 + \frac{D}{\sqrt{M}})^k)$ , by virtue of (10) we can emit all the result tuples in  $\mathcal{Q}$  using  $O((1 + \frac{D}{\sqrt{M}})^k \cdot \frac{M}{B})$  I/Os.

It remains to explain how to ensure that  $R_e(I_i, I_j)$  can always be loaded into memory using  $O(M/B)$  I/Os. This requires only  $O((1 + \frac{D}{\sqrt{M}})^2 \cdot \frac{1}{B} + \text{sort}(N))$  extra I/Os to prepare the input relations of  $\mathcal{Q}$  appropriately. The details are standard and moved to Appendix A. With that, we conclude the proof of Lemma 5.

## 4 Batched Two-Way Semi-Join Reductions

This section will study an interesting variant of the traditional “semi-join” problem. This variant is extracted from a sub-problem that will arise in Section 5 when we discuss how to fulfill the objective outlined in (9).

We are given a binary relation  $R$  with schema  $\{X, Y\}$ , together with  $\ell$  unary relations  $S_1, S_2, \dots, S_\ell$  with schema  $\{X\}$  and another  $\ell$  unary relations  $T_1, T_2, \dots, T_\ell$  with schema  $\{Y\}$ . Both attributes  $X$  and  $Y$  have degrees at most  $\lambda$  in  $R$ , whereas every  $S_i$  and every  $T_i$  ( $i \in [\ell]$ ) have been sorted by  $X$  and  $Y$ , respectively. For each  $i \in [\ell]$ , define a join

$$Q_i = \{R, S_i, T_i\} \quad (11)$$

whose result  $join(Q_i)$  is a subset of  $R$ , including every tuple  $\mathbf{u} \in R$  with  $\mathbf{u}(X) \in S_i$  and  $\mathbf{u}(Y) \in T_i$ . Our objective is to compute, for every  $i \in [\ell]$ , the result  $join(Q_i)$  and write it to consecutive blocks on the disk. We will refer to the above as the *batched two-way semi-join reduction problem*.

The problem admits a “textbook solution” that computes each  $join(Q_i)$  individually as follows. Specifically, we can first sort  $R$  on attribute  $X$  and compute  $R' = R \times S_i$  by scanning  $R$  and  $S_i$  synchronously. Then, we sort  $R'$  on attribute  $Y$  and compute  $R'' = R' \times T_i$  by scanning  $R'$  and  $T_i$  synchronously. The relation  $R''$  is the join result  $join(Q_i)$ . Executing these steps for each  $i \in [\ell]$  necessitates a total I/O cost of  $O(\ell \cdot \text{sort}(|R|) + \sum_{i=1}^{\ell} \lceil (|S_i| + |T_i|)/B \rceil)$ . On the other hand, we will prove:

► **Lemma 6.** *Batched two-way semi-join reduction can be solved with an I/O complexity*

$$O\left(\left(\frac{|R|}{\lambda} + \frac{\lambda}{M}\right) \frac{\sum_{i=1}^{\ell} |S_i| + |T_i|}{B} + \ell \cdot \left\lceil \frac{|R|}{B} \right\rceil + \frac{\ell \cdot |R|^2 M}{\lambda^2 \cdot B} + \text{sort}(|R|)\right).$$

*The above holds for arbitrary integers  $\lambda$  and  $\ell$  (which may not be constants).*

The lemma is most interesting under  $\lambda = \sqrt{|R| \cdot M}$ , in which case the I/O complexity can be simplified to  $O(\sqrt{|R|/M} \cdot (\sum_{i=1}^{\ell} |S_i| + |T_i|)/B + \ell \cdot |R|/B + \text{sort}(|R|))$ . In the “lopsided situation” where  $\sum_{i=1}^{\ell} |S_i| + |T_i| \leq \ell \sqrt{|R| \cdot M}$  – that is, on average each unary relation has a size  $O(\sqrt{|R| \cdot M}) = O(\lambda)$  – the I/O complexity becomes  $O(\ell \cdot \lceil |R|/B \rceil + \text{sort}(|R|))$ , which improves the aforementioned textbook solution by a factor of  $O(\log_{M/B}(|R|/B))$  for large  $\ell$ .

The rest of the section serves as a proof of Lemma 6. We start by partitioning the integer domain  $\mathbb{N}$  into a set  $S_X^{intv}$  of intervals such that

- $|S_X^{intv}| = O(|R|/\lambda)$  and
- for each interval  $I \in S_X^{intv}$ , the relation  $R$  has  $O(\lambda)$  tuples  $\mathbf{u}$  with  $\mathbf{u}(X) \in I$ .

Symmetrically, obtain a set  $S_Y^{intv}$  of intervals satisfying two analogous conditions with respect to  $Y$ . The intervals in  $S_X^{intv}$  and  $S_Y^{intv}$  can be obtained in  $O(\text{sort}(|R|))$  I/Os<sup>3</sup>.

<sup>3</sup> We will explain this only for  $S_X^{intv}$  due to symmetry. Sort and group the tuples of  $R$  by their  $X$ -values. Each group has a size of at most  $\lambda$ . Next, we will scan the groups in ascending order of their  $X$ -values and, in doing so, divide the tuples of  $R$  using special tokens. First, place a token before the first group and then start the scan. Every time an entire group of tuples has been scanned, place another token at the end of the group if we have seen at least  $\lambda$  tuples since the last token. Finally, place another token at the end of the last group. It is easy to see that there can be at most  $2\lambda$  tuples between any two consecutive tokens and the number of tokens is  $O(|R|/\lambda)$ . The desired set  $S_X^{intv}$  of intervals can then be easily determined based on the tokens.

## 21:10 Subgraph Enumeration in Optimal I/O Complexity

For any interval  $I_X \in S_X^{intv}$ , any interval  $I_Y \in S_Y^{intv}$ , and any  $i \in [\ell]$ , define

$$\begin{aligned} R(I_X, I_Y) &= \{\mathbf{u} \in R \mid \mathbf{u}(X) \in I_X \text{ and } \mathbf{u}(Y) \in I_Y\} \\ S_i(I_X) &= \{\mathbf{u} \in X \mid \mathbf{u}(X) \in I_X\} \\ T_i(I_Y) &= \{\mathbf{u} \in Y \mid \mathbf{u}(Y) \in I_Y\}. \end{aligned}$$

After  $O(\ell + \frac{|R|^2}{\lambda^2 \cdot B} + \text{sort}(|R|) + \frac{\ell \cdot |R|}{\lambda B} + \sum_{i=1}^{\ell} (|S_i| + |T_i|)/B)$  I/Os, we can store each

- $R(I_X, I_Y)$  in consecutive blocks whose starting address can be located in one I/O;
- $S_i(I_X)$  in consecutive blocks whose starting address can be located in one I/O;
- $T_i(I_Y)$  in consecutive blocks whose starting address can be located in one I/O.

The details are standard and moved to Appendix B.

For each  $i \in [\ell]$ , we initialize an empty disk file for  $\mathcal{Q}_i$  and will gradually populate the file with tuples of  $\text{join}(\mathcal{Q}_i)$  until eventually the file's content is exactly  $\text{join}(\mathcal{Q}_i)$ . Motivated by the fact

$$\text{join}(\mathcal{Q}_i) = \bigcup_{(I_X, I_Y) \in S_X^{intv} \times S_Y^{intv}} R(I_X, I_Y) \bowtie S_i(I_X) \bowtie T_i(I_Y) \quad (12)$$

we process each interval pair  $(I_X, I_Y) \in S_X^{intv} \times S_Y^{intv}$  separately and, in doing so, append  $R(I_X, I_Y) \bowtie S_i(I_X) \bowtie T_i(I_Y)$  to the file of  $\mathcal{Q}_i$  for every  $i \in [\ell]$ .

We now elaborate how to process a pair  $(I_X, I_Y) \in S_X^{intv} \times S_Y^{intv}$ . Conceptually, partition  $R(I_X, I_Y)$  (arbitrarily) into *chunks*, each of which contains  $M - B \geq M/2$  tuples except possibly the last chunk. For each chunk, we first read it into memory using  $O(M/B)$  I/Os; let  $R_{\text{chunk}}$  be the set of tuples in that chunk. Keeping  $R_{\text{chunk}}$  in memory, we then – for each  $i \in [\ell]$  in turn – compute and append  $R_{\text{chunk}} \bowtie S_i(I_X) \bowtie T_i(I_Y)$  to the file of  $\mathcal{Q}_i$  using  $O(\frac{|S_i(I_X)| + |T_i(I_Y)| + M}{B})$  I/Os. For this purpose, we scan  $S_i$  in its entirety using one block of memory. As soon as a tuple  $\mathbf{v} \in S_i(I_X)$  is brought into memory, it is compared to all the tuples in  $R_{\text{chunk}}$ . In doing so, for each tuple  $\mathbf{u} \in R_{\text{chunk}}$ , we track whether any tuple  $\mathbf{v} \in S_i(I_X)$  with  $\mathbf{u}(X) = \mathbf{v}(X)$  has been seen; if so, we “mark”  $\mathbf{u}$ . Similarly, by scanning  $T_i(I_Y)$  once, we can detect, for each tuple  $\mathbf{u} \in R_{\text{chunk}}$ , whether  $T_i(I_Y)$  has any tuple  $\mathbf{v}$  with  $\mathbf{u}(Y) = \mathbf{v}(Y)$ ; if so, we “mark”  $\mathbf{u}$ . The tuples of  $R_{\text{chunk}} \bowtie S_i(I_X) \bowtie T_i(I_Y)$  are exactly those in  $R_{\text{chunk}}$  that receive two marks (one from scanning  $S_i(I_X)$  and the other from  $T_i(I_Y)$ ). These tuples are then written to the disk in  $O(M/B)$  I/Os. In the entire processing of  $(I_X, I_Y)$ ,  $S_i(I_X)$  and  $T_i(I_Y)$  are scanned  $O(\lceil |R(I_X, I_Y)|/M \rceil)$  times, i.e., the number of chunks.

By executing the above algorithm for every  $(I_X, I_Y)$ , we produce the result of  $\mathcal{Q}_i$  on the disk for all  $i \in [\ell]$  with an I/O complexity:

$$O\left(\sum_{(I_X, I_Y) \in S_X^{intv} \times S_Y^{intv}} \left\lceil \frac{|R(I_X, I_Y)|}{M} \right\rceil \cdot \sum_{i=1}^{\ell} \frac{|S_i(I_X)| + |T_i(I_Y)| + M}{B}\right) \quad (13)$$

In Appendix C, we show how to relate the above to the degree threshold  $\lambda$  and prove

$$(13) = O\left(\left(\frac{|R|}{\lambda} + \frac{\lambda}{M}\right) \frac{\sum_{i=1}^{\ell} |S_i| + |T_i|}{B} + \frac{\ell \cdot |R|}{B} + \frac{\ell \cdot |R|^2 M}{\lambda^2 \cdot B}\right). \quad (14)$$

With this, we can then conclude the proof of Lemma 6.

## 5 I/O-Efficient Heavy-Light Decomposition

We now return to processing a schema-clean join consisting purely of binary relations and will accomplish the mission described in (9). This section effectively proves the lemma below.

► **Lemma 7.** *Let  $\mathcal{Q}$  be a schema-clean join on at least two binary relations, whose schema graph is  $\mathcal{G} = (\mathcal{X}, \mathcal{E})$ . Let  $k = |\mathcal{X}|$ , and let  $N$  be the input size of  $\mathcal{Q}$ . Fix an arbitrary  $\mathcal{H} \subseteq \mathcal{X}$ . In  $T_{\mathcal{H}} = O(N^{k/2}/(M^{k/2-1}B))$  I/Os, we can produce*

- *config( $\mathcal{H}$ ) in consecutive disk blocks;*
- *each input relation of  $\mathcal{Q}''(\boldsymbol{\eta})$  in consecutive disk blocks for every active configuration  $\boldsymbol{\eta} \in \text{config}(\mathcal{H})$ .*

Because  $\mathcal{X}$  has  $2^k = O(1)$  subsets  $\mathcal{H}$ , the above lemma indicates  $\sum_{\mathcal{H} \subseteq \mathcal{X}} T_{\mathcal{H}} = O(N^{k/2}/(M^{k/2-1}B))$ , as needed in (9). We will explain in Section 5.1 how to generate  $\text{config}(\mathcal{H})$  and then in Section 5.2 how to create the input relations of the simplified residual joins  $\mathcal{Q}''(\boldsymbol{\eta})$  for all  $\boldsymbol{\eta} \in \text{config}(\mathcal{H})$ . As will be clear later, the core of the former (resp., latter) task boils down to the problem tackled in Section 3 (resp., 4).

**Preprocessing.** For each attribute  $X \in \mathcal{X}$ , we use  $\mathbf{adom}_{\mathcal{Q}}^{\text{H}}(X)$  (resp.,  $\mathbf{adom}_{\mathcal{Q}}^{\text{L}}(X)$ ) to represent the set of heavy (resp., light) values in  $\mathbf{adom}_{\mathcal{Q}}(X)$ . Recall that, for each hyperedge  $e = \{X, Y\} \in \mathcal{E}$ , the symbol  $R_e$  denotes the only relation in  $\mathcal{Q}$  whose schema is  $e$ . We define:

$$R_e^{\text{HH}} = \{\mathbf{u} \in R_e \mid \mathbf{u}(X) \in \mathbf{adom}_{\mathcal{Q}}^{\text{H}}(X) \text{ and } \mathbf{u}(Y) \in \mathbf{adom}_{\mathcal{Q}}^{\text{H}}(Y)\}. \quad (15)$$

$$R_e^{\text{LL}} = \{\mathbf{u} \in R_e \mid \mathbf{u}(X) \in \mathbf{adom}_{\mathcal{Q}}^{\text{L}}(X) \text{ and } \mathbf{u}(Y) \in \mathbf{adom}_{\mathcal{Q}}^{\text{L}}(Y)\}. \quad (16)$$

If  $e$  contains a heavy attribute  $X \in \mathcal{H}$  and a light attribute  $Y \in \mathcal{L}$  (recall:  $\mathcal{L} = \mathcal{X} \setminus \mathcal{H}$ ), we refer to  $e \in \mathcal{E}$  as a *crossing hyperedge*. Given a crossing hyperedge  $e$ , we define:

$$R_e^{\text{HL}} = \{\mathbf{u} \in R_e \mid \mathbf{u}(X) \in \mathbf{adom}_{\mathcal{Q}}^{\text{H}}(X) \text{ and } \mathbf{u}(Y) \in \mathbf{adom}_{\mathcal{Q}}^{\text{L}}(Y)\}. \quad (17)$$

It is rudimentary to use  $\text{sort}(N)$  I/Os to produce on the disk:

- $\mathbf{adom}_{\mathcal{Q}}^{\text{H}}(X)$  in sorted order for every  $X \in \mathcal{X}$ ;
- $\mathbf{adom}_{\mathcal{Q}}^{\text{L}}(X)$  in sorted order for every  $X \in \mathcal{X}$ ;
- $R_e^{\text{HH}}$  for every  $e \in \mathcal{E}$ ;
- $R_e^{\text{LL}}$  for every  $e \in \mathcal{E}$ ;
- $R_e^{\text{HL}}$  for every crossing hyperedge  $e \in \mathcal{E}$ .

We remark that  $\text{sort}(N) = O(\frac{N}{B} \log_{M/B} \frac{N}{B}) = O(\frac{N}{B} \log_{M/B} \frac{N}{M}) = O(\frac{N}{B} \sqrt{\frac{N}{M}}) = O(\frac{N^{k/2}}{M^{k/2-1}B})$ , where the last step used the fact  $k \geq 3$  (because  $\mathcal{Q}$  has at least two relations and is schema-clean). In other words, sorting is within the target budget of Lemma 7.

### 5.1 Generating $\text{config}(\mathcal{H})$

Let us first explain how to compute  $\text{config}(\mathcal{H})$  under the condition  $\mathcal{H} = \mathcal{X}$ . In such case, we construct a join

$$\mathcal{Q}_{\mathcal{H}} = \{R_e^{\text{HH}} \mid e \in \mathcal{E}\}.$$

Note that  $\text{config}(\mathcal{H})$  is precisely the result  $\text{join}(\mathcal{Q}_{\mathcal{H}})$  of  $\mathcal{Q}_{\mathcal{H}}$ . The input relations of  $\mathcal{Q}_{\mathcal{H}}$  contain only values classified as “heavy” for the original join  $\mathcal{Q}$ . Hence, for every attribute  $X \in \mathcal{X}$ , it holds that  $\mathbf{adom}_{\mathcal{Q}_{\mathcal{H}}}(X)$  – the active domain of  $X$  under  $\mathcal{Q}_{\mathcal{H}}$  – has size  $O(N/\lambda)$ , which is  $O(\sqrt{N/M})$  given the heavy-value threshold  $\lambda$  in (1). We compute  $\text{join}(\mathcal{Q}_{\mathcal{H}})$  – namely,  $\text{config}(\mathcal{H})$  – using Lemma 5 (plugging in  $D = O(\sqrt{N/M})$ ). The I/O cost is

$$O\left(\left(1 + \frac{\sqrt{N/M}}{\sqrt{M}}\right)^k \cdot \frac{M}{B} + \text{sort}(N)\right) + O\left(\frac{|\text{config}(\mathcal{H})|}{B}\right) \quad (18)$$

where the term  $O(|\text{config}(\mathcal{H})|/B)$  does not come from Lemma 5, but is instead due to the need of writing  $\text{config}(\mathcal{H})$  to the disk. Applying the relationship  $|\text{config}(\mathcal{H})| = O((N/M)^{|\mathcal{H}|/2}) = O((N/M)^{k/2})$ , the reader can easily confirm that (18) is bounded by  $O(N^{k/2}/(M^{k/2-1}B))$ .

The case where  $\mathcal{H} \subset \mathcal{X}$  can be dealt with in a more conventional manner:

1.  $CPadom^{\mathcal{H}} = \times_{X \in \mathcal{H}} \mathbf{adom}_{\mathcal{Q}}^{\mathcal{H}}(X)$ , i.e., the cartesian product of the heavy-value sets  $\mathbf{adom}_{\mathcal{Q}}^{\mathcal{H}}(X)$  for all the heavy attributes  $X \in \mathcal{H}$ .
2.  $R^* = CPadom^{\mathcal{H}}$   
/\* henceforth, we will regard  $R^*$  as a relation with schema  $\mathcal{H}^*/$
3. **for** each hyperedge  $e \in \mathcal{E}$  such that  $e \subseteq \mathcal{H}$  **do**
4.  $R^* = R^* \times R_e$  (semi-join)

The final  $R^*$  is the  $\text{config}(\mathcal{H})$  we aim to compute. Regarding the cost, Line 1 can be implemented<sup>4</sup> in  $O(|CPadom^{\mathcal{H}}|/B)$  I/Os (dominated by the cost of writing  $CPadom^{\mathcal{H}}$  to the disk), where  $|CPadom^{\mathcal{H}}| = O((\sqrt{N/M})^{|\mathcal{H}|})$ . With sorting, we can perform Lines 2-4 in  $O(\text{sort}(|CPadom^{\mathcal{H}}|) + \text{sort}(N))$  I/Os. The total overhead is thus bounded by

$$O\left(\left(\frac{N}{M}\right)^{|\mathcal{H}|/2} \log_{M/B} \frac{N}{M} + \text{sort}(N)\right) = O\left(\left(\frac{N}{M}\right)^{k/2} + \text{sort}(N)\right) = O(N^{k/2}/(M^{k/2-1}B)) \quad (19)$$

I/Os, where the first equality holds because  $|\mathcal{H}| < |\mathcal{X}| = k$ .

## 5.2 Generating the Input Relations for the Simplified Residual Joins

We now proceed to explain how to create the input relations of  $\mathcal{Q}''(\boldsymbol{\eta})$  for every active configuration  $\boldsymbol{\eta} \in \text{config}(\mathcal{H})$ . It suffices to consider  $\mathcal{H} \subset \mathcal{X}$  (if  $\mathcal{H} = \mathcal{X}$ , every  $\mathcal{Q}''(\boldsymbol{\eta})$  is empty, i.e., there are no input relations at all). The relationship  $\mathcal{H} \subset \mathcal{X}$  implies a useful property:  $O(\text{sort}(|\text{config}(\mathcal{H})|)) = O(\text{sort}((N/M)^{|\mathcal{H}|/2})) = O(N^{k/2}/(M^{k/2-1}B))$ , following the same derivation as in (19).

**Input Relations of Residual Joins.** We start by generating  $R_e(\boldsymbol{\eta})$  – the subset of the relation  $R_e$  that “agrees” with  $\boldsymbol{\eta}$ ; see definition in Section 2.2 – for every hyperedge  $e \in \mathcal{E}$  relevant to  $\mathcal{H}$  (i.e.,  $e$  is not subsumed by  $\mathcal{H}$ , as defined in Section 2.2) and active configuration  $\boldsymbol{\eta} \in \text{config}(\mathcal{H})$ . This is, in fact, trivial if  $e$  is disjoint with  $\mathcal{H}$  (i.e., both attributes in  $e$  are light), in which case  $R_e(\boldsymbol{\eta})$  is simply the relation  $R_e^{\text{LL}}$  in (16), which has already been obtained.

It remains to consider the crossing hyperedges  $e \in \mathcal{E}$  (which contain one heavy attribute and one light attribute). W.l.o.g., let  $e = \{X, Y\}$  where  $X \in \mathcal{H}$  and  $Y \in \mathcal{L}$ . Imagine dividing  $R_e^{\text{HL}}$  into groups according to the  $X$ -values of the tuples therein. Then, for each active configuration  $\boldsymbol{\eta} \in \text{config}(\mathcal{H})$ , the set  $R_e(\boldsymbol{\eta})$  is exactly the group having the  $X$ -value  $\boldsymbol{\eta}(X)$ . Motivated by this, we sort  $R_e^{\text{HL}}$  (defined in (17)) by attribute  $X$  in  $O(\text{sort}(N))$  I/Os and sort the active configurations  $\boldsymbol{\eta} \in \text{config}(\mathcal{H})$  by  $\boldsymbol{\eta}(X)$  in  $O(\text{sort}(|\text{config}(\mathcal{H})|)) = O(N^{k/2}/(M^{k/2-1}B))$  I/Os. Then, by going through  $R_e^{\text{HL}}$  and  $\text{config}(\mathcal{H})$  synchronously once, we can, for each  $\boldsymbol{\eta} \in \text{config}(\mathcal{H})$ , identify the starting disk address of  $R_e(\boldsymbol{\eta})$  and store this address along with  $\boldsymbol{\eta}$ .

<sup>4</sup> Using the textbook algorithm “blocked nested loop”.

Now that we have created  $R_e(\boldsymbol{\eta})$  on the disk for every relevant hyperedge  $e \in \mathcal{E}$  and active configuration  $\boldsymbol{\eta} \in \text{config}(\mathcal{H})$ , we can acquire the residual relation  $R'_e(\boldsymbol{\eta})$  via a projection, as indicated in (2), which requires only a single scan of  $R_e(\boldsymbol{\eta})$ . This generates the input relations of the residual joins  $\mathcal{Q}'(\boldsymbol{\eta})$  of all  $\boldsymbol{\eta} \in \text{config}(\mathcal{H})$ . The overall I/O cost is bounded by  $O(N^{k/2}/(M^{k/2-1}B))$ .

While the above algorithm fulfills the objective of creating the input relations of all residual joins, we will modify it slightly to better facilitate the subsequent algorithm design. The purpose of these modifications is to ensure that, for every active configuration  $\boldsymbol{\eta} \in \text{config}(\mathcal{H})$  and every hyperedge  $e \in \mathcal{E}$  containing a border attribute<sup>5</sup>  $Y \in \mathcal{L}$ , there should be a copy of  $R'_e(\boldsymbol{\eta})$  on the disk that has been sorted on attribute  $Y$ . To that end, we change the aforementioned method for computing  $R_e(\boldsymbol{\eta})$  as follows:

- If  $e$  is disjoint with  $\mathcal{H}$ , sort  $R_e^{\text{LL}}$  on  $Y$  in  $O(\text{sort}(N))$  I/Os. Remember that  $R_e^{\text{LL}} = R_e(\boldsymbol{\eta})$  regardless of  $\boldsymbol{\eta} \in \text{config}(\mathcal{H})$ .
- If  $e$  is a crossing hyperedge of the form  $\{X, Y\}$  where  $X \in \mathcal{H}$ , instead of sorting  $R_e^{\text{HL}}$  only by attribute  $X$  as described earlier, we perform the sorting first by  $X$  and then break ties by  $Y$ . After that, the relations  $R_e(\boldsymbol{\eta})$  of all  $\boldsymbol{\eta} \in \text{config}(\mathcal{H})$  can still be obtained by going through (the sorted)  $R_e^{\text{HL}}$  and (the sorted)  $\text{config}(\mathcal{H})$  synchronously once, but the tuples of each  $R_e(\boldsymbol{\eta})$  are now sorted on attribute  $Y$ .

Every relation  $R'_e(\boldsymbol{\eta})$  is still computed by taking a projection of  $R_e(\boldsymbol{\eta})$  as before. The overall I/O complexity remains as  $O(N^{k/2}/(M^{k/2-1}B))$ .

**Input Relations of Simplified Residual Joins.** Next, for each border attribute  $X \in \mathcal{X}$  and every active configuration  $\boldsymbol{\eta} \in \text{config}(\mathcal{H})$ , we compute the set  $R''_X(\boldsymbol{\eta})$  defined in (6). This is the intersection of all the  $\Pi_X(R'_e(\boldsymbol{\eta}))$ , where  $e$  ranges over all the hyperedges in  $\mathcal{G}'$  (defined in Section 2.2) containing  $X$ . Since (i) we have prepared a copy of  $R'_e(\boldsymbol{\eta})$  sorted by  $X$  on the disk and (ii)  $R'_e(\boldsymbol{\eta})$  has a size at most  $N_\boldsymbol{\eta}$  (see (4)), the intersection can be computed in  $O(\lceil N_\boldsymbol{\eta}/B \rceil)$  I/Os; note that the  $R''_X(\boldsymbol{\eta})$  thus computed is sorted on  $X$  and has a size at most  $N_\boldsymbol{\eta}$ . The total cost for obtaining the sets  $R''_X(\boldsymbol{\eta})$  of all  $\boldsymbol{\eta} \in \text{config}(\mathcal{H})$  is bounded by

$$\sum_{\boldsymbol{\eta} \in \text{config}(\mathcal{H})} O(\lceil N_\boldsymbol{\eta}/B \rceil) = |\text{config}(\mathcal{H})| + \sum_{\boldsymbol{\eta} \in \text{config}(\mathcal{H})} O(N_\boldsymbol{\eta}/B) \quad (20)$$

Deng et al. showed [12, Lemma 11] that

$$\sum_{\boldsymbol{\eta} \in \text{config}(\mathcal{H})} N_\boldsymbol{\eta} = O(N^{k/2}/M^{k/2-1}).$$

Therefore:

$$(20) = O(N^{k/2}/M^{k/2}) + O(N^{k/2}/(M^{k/2-1}B)) = O(N^{k/2}/(M^{k/2-1}B)). \quad (21)$$

In the scenario where every hyperedge  $e \in \mathcal{E}$  contains at most one light attribute, the set  $\mathcal{I}$  of isolated attributes is exactly the set of border attributes, and hence, the simplified residual join  $\mathcal{Q}''(\boldsymbol{\eta})$  of each  $\boldsymbol{\eta} \in \text{config}(\mathcal{H})$  – see (7) – consists of  $\{R''_X(\boldsymbol{\eta}) \mid X \text{ is a border attribute}\}$ . We are therefore done with generating the input relations of all the simplified residual joins.

In the final segment of our proof (for Lemma 7), we consider that  $\mathcal{E}$  has at least one edge  $e = \{X_1, X_2\}$  where both  $X_1$  and  $X_2$  are light attributes. Note that this means  $|\mathcal{H}| \leq k - 2$ . Next, given an arbitrary such hyperedge  $e = \{X_1, X_2\}$ , we will explain how to produce on the disk the simplified residual relations  $R''_e(\boldsymbol{\eta})$  under all active configurations  $\boldsymbol{\eta} \in \text{config}(\mathcal{H})$  with  $O(N^{k/2}/(M^{k/2-1}B))$  I/Os in total.

<sup>5</sup> As defined in Section 2.2, a border attribute is a light attribute that co-appears with a heavy attribute in some hyperedge of  $\mathcal{E}$ .



## 21:14 Subgraph Enumeration in Optimal I/O Complexity

This is trivial if neither  $X_1$  nor  $X_2$  is a border attribute – in this case,  $R_e''(\boldsymbol{\eta}) = R_e'(\boldsymbol{\eta})$ , which has already been computed. Now consider the case where  $e$  has only one border attribute, say,  $X_1$  (thus  $X_2$  is not a border attribute; nonetheless, remember that  $X_2$  is a light attribute). Earlier we have prepared a copy of  $R_e'(\boldsymbol{\eta}) = R_e^{\text{LL}}$  sorted on  $X_1$ , as well as  $R_{X_1}''(\boldsymbol{\eta})$ , which is also sorted on  $X_1$ . Because the sizes of both  $R_e'(\boldsymbol{\eta})$  and  $R_{X_1}''(\boldsymbol{\eta})$  are at most  $N_{\boldsymbol{\eta}}$ , we can compute  $R_e''(\boldsymbol{\eta}) = R_e'(\boldsymbol{\eta}) \bowtie R_{X_1}''(\boldsymbol{\eta})$  in  $O(\lceil N_{\boldsymbol{\eta}}/B \rceil)$  I/Os. Hence, the simplified residual relations  $R_e''(\boldsymbol{\eta})$  under all  $\boldsymbol{\eta} \in \text{config}(\mathcal{H})$  can be obtained in  $O(\sum_{\boldsymbol{\eta} \in \text{config}(\mathcal{H})} O(\lceil N_{\boldsymbol{\eta}}/B \rceil))$  I/Os, which is  $O(N^{k/2}/(M^{k/2-1}B))$ ; see (20) and (21).

The most non-trivial situation arises when both  $X_1$  and  $X_2$  in  $e$  are border attributes. This is where the “batched two-way semi-join reductions” problem (we will abbreviate the problem name as “batched semi” from now on) discussed in Section 4 comes in. Let us construct an instance of that problem as follows:

- The value of  $\ell$  in “batched semi” equals  $|\text{config}(\mathcal{H})| = O((N/M)^{|\mathcal{H}|/2}) = O((N/M)^{k/2-1})$ , applying  $|\mathcal{H}| \leq k - 2$ .
- The binary relation  $R$  in “batched semi” corresponds to  $R_e^{\text{LL}}$ .
- The degree threshold  $\lambda$  in “batched semi” equals the value of  $\lambda$  in (1), which is  $O(\sqrt{NM})$ .
- The  $i$ -th ( $i \in [\ell]$ ) unary relation  $S_i$  in “batched semi” corresponds to the set  $R_{X_1}''(\boldsymbol{\eta})$  of the  $i$ -th<sup>6</sup> active configuration  $\boldsymbol{\eta} \in \text{config}(\mathcal{H})$ . Remember that  $R_{X_1}''(\boldsymbol{\eta})$  has been sorted on  $X_1$ .
- The  $i$ -th ( $i \in [\ell]$ ) unary relation  $T_i$  in “batched semi” corresponds to set  $R_{X_2}''(\boldsymbol{\eta})$  of the  $i$ -th active configuration  $\boldsymbol{\eta} \in \text{config}(\mathcal{H})$ . Remember that  $R_{X_2}''(\boldsymbol{\eta})$  has been sorted on  $X_2$ .

“Batched semi” computes the result  $\text{join}(\mathcal{Q}_i)$  of the join  $\mathcal{Q}_i$  defined in (11). Based on our construction,  $\text{join}(\mathcal{Q}_i)$  equals  $R_e^{\text{LL}} \bowtie R_{X_1}''(\boldsymbol{\eta}) \bowtie R_{X_2}''(\boldsymbol{\eta})$  – with  $\boldsymbol{\eta}$  being the  $i$ -th configuration in  $\text{config}(\mathcal{H})$  – which is exactly the simplified residual relation  $R_e''(\boldsymbol{\eta})$  of  $e$  under  $\boldsymbol{\eta}$ . From Lemma 6, we can assert that the relations  $R_e''(\boldsymbol{\eta})$  under all  $\boldsymbol{\eta} \in \text{config}(\mathcal{H})$  can be computed with an I/O cost

$$\begin{aligned}
& O\left(\left(\frac{N}{\lambda} + \frac{\lambda}{M}\right) \frac{\sum_{\boldsymbol{\eta} \in \text{config}(\mathcal{H})} (|R_{X_1}''(\boldsymbol{\eta})| + |R_{X_2}''(\boldsymbol{\eta})|)}{B} + \frac{\ell \cdot N}{B} + \frac{\ell \cdot N^2 M}{\lambda^2 \cdot B} + \text{sort}(N)\right) \\
&= O\left(\frac{\sqrt{N}}{\sqrt{MB}} \left(\sum_{\boldsymbol{\eta} \in \text{config}(\mathcal{H})} (|R_{X_1}''(\boldsymbol{\eta})| + |R_{X_2}''(\boldsymbol{\eta})|)\right) + \frac{\ell \cdot N}{B} + \text{sort}(N)\right) \\
&= O\left(\frac{\sqrt{N}}{\sqrt{MB}} \left(\sum_{\boldsymbol{\eta} \in \text{config}(\mathcal{H})} (|R_{X_1}''(\boldsymbol{\eta})| + |R_{X_2}''(\boldsymbol{\eta})|)\right) + \frac{N^{k/2}}{M^{k/2-1}B}\right) \tag{22}
\end{aligned}$$

where the last step applied  $\ell = O((N/M)^{k/2-1})$ .

► **Proposition 8.**  $\sum_{\boldsymbol{\eta} \in \text{config}(\mathcal{H})} |R_{X_1}''(\boldsymbol{\eta})| = O\left(\sqrt{NM} \cdot \left(\frac{N}{M}\right)^{k/2-1}\right)$

**Proof.** Because  $X_1$  is a border attribute, there must exist a heavy attribute  $H \in \mathcal{H}$  such that  $e = \{X_1, Y\}$  is a hyperedge in  $\mathcal{E}$ . By how  $R_{X_1}''(\boldsymbol{\eta})$  is computed in (6), we know that  $R_{X_1}''(\boldsymbol{\eta})$  is a subset of  $R_e'(\boldsymbol{\eta})$ . Furthermore, since  $R_e'(\boldsymbol{\eta}) = \Pi_{e \setminus \mathcal{H}}(R_e(\boldsymbol{\eta}))$  (see (2)), it must hold that

$$|R_{X_1}''(\boldsymbol{\eta})| \leq |R_e'(\boldsymbol{\eta})| \leq |R_e(\boldsymbol{\eta})|.$$

<sup>6</sup> Here, impose an arbitrary ordering of  $\text{config}(\mathcal{H})$ .



Next, we will prove that

$$\sum_{\boldsymbol{\eta} \in \text{config}(\mathcal{H})} |R_e(\boldsymbol{\eta})| = O\left(\sqrt{NM} \cdot \left(\frac{N}{M}\right)^{k/2-1}\right) \quad (23)$$

which will then imply the claim in the proposition.

To that end, we break the left hand side of (23) as follows:

$$\sum_{\boldsymbol{\eta} \in \text{config}(\mathcal{H})} |R_e(\boldsymbol{\eta})| = \sum_{y \in \text{adom}^H(Y)} \left( \sum_{\boldsymbol{\eta} \in \text{config}(\mathcal{H}): \boldsymbol{\eta}(Y)=y} |R_e(\boldsymbol{\eta})| \right). \quad (24)$$

Observe from the definition of  $R_e(\boldsymbol{\eta})$  that  $R_e(\boldsymbol{\eta})$  is simply  $\sigma_{Y=y}(R_e)$  where  $\sigma$  is the standard *selection* operator in relational algebra. Hence, for any  $y \in \text{adom}^H(Y)$ , the size  $|R_e(\boldsymbol{\eta})|$  is the same for all  $\boldsymbol{\eta} \in \text{config}(\mathcal{H})$  satisfying  $\boldsymbol{\eta}(Y) = y$ . As there are at most  $O((N/M)^{(|\mathcal{H}|-1)/2})$  such active configurations  $\boldsymbol{\eta}$ , we can derive

$$\begin{aligned} (24) &= O((N/M)^{(|\mathcal{H}|-1)/2}) \cdot \sum_{y \in \text{adom}^H(Y)} |\sigma_{Y=y}(R_e)| \\ &\leq O((N/M)^{(|\mathcal{H}|-1)/2}) \cdot |R_e| \\ &= O\left(\frac{N^{(|\mathcal{H}|-1)/2}}{M^{(|\mathcal{H}|-1)/2}}\right) \\ (\text{as } |H| \leq k-2) &= O\left(\sqrt{NM} \cdot \left(\frac{N}{M}\right)^{k/2-1}\right) \end{aligned}$$

as claimed in (23). ◀

A symmetric argument yields  $\sum_{\boldsymbol{\eta} \in \text{config}(\mathcal{H})} |R''_{X_2}(\boldsymbol{\eta})| = O(\sqrt{NM} \cdot (\frac{N}{M})^{k/2-1})$ . Plugging these relationships into (22), we can derive

$$(22) = O\left(\frac{\sqrt{N}}{\sqrt{MB}} \cdot \sqrt{NM} \left(\frac{N}{M}\right)^{k/2-1} + \frac{N^{k/2}}{M^{k/2-1}B}\right) = O\left(\frac{N^{k/2}}{M^{k/2-1}B}\right).$$

We now conclude that the input relations of the simplified residual joins  $\mathcal{Q}''(\boldsymbol{\eta})$  of all  $\boldsymbol{\eta} \in \text{config}(\mathcal{H})$  can be computed in  $O(N^{k/2}/(M^{k/2-1}B))$  I/Os in total. This completes the entire proof of Lemma 7.

## 6 Conclusions

By putting together Lemmas 1, 4, and 7, we have arrived at the main result of this paper:

► **Theorem 9.** *Let  $G = (V, E)$  be a simple undirected graph with no isolated vertices. Let  $Q = (V_Q, E_Q)$  be a simple undirected connected pattern graph. There is an algorithm in the external memory model that, with probability at least  $1 - 1/|E|^c$  where  $c$  is an arbitrarily high constant decided before running the algorithm, emits every occurrence of  $Q$  in  $G$  exactly once with  $O(|E|^\rho/(M^{\rho-1}B))$  I/Os, where  $\rho$  is the fractional edge covering number of  $Q$ ,  $M$  is the number of words in memory, and  $B$  is the number of words in a disk block. The same I/O complexity holds also in expectation.*

The theorem optimally settles the subgraph enumeration problem in external memory when randomization is permitted. The main open problem left behind by this work is to achieve the I/O complexity  $O(|E|^\rho/(M^{\rho-1}B))$  deterministically.

## References

- 1 Alok Aggarwal and Jeffrey Scott Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM (CACM)*, 31(9):1116–1127, 1988. doi:10.1145/48529.48535.
- 2 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM (JACM)*, 42(4):844–856, 1995. doi:10.1145/210332.210337.
- 3 Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997. doi:10.1007/BF02523189.
- 4 Kaleb Alway, Eric Blais, and Semih Salihoglu. Box covers and domain orderings for beyond worst-case join processing. In *Proceedings of International Conference on Database Theory (ICDT)*, pages 3:1–3:23, 2021. doi:10.4230/LIPIcs.ICDT.2021.3.
- 5 Suman K. Bera, Noujan Pashanasangi, and C. Seshadhri. Near-linear time homomorphism counting in bounded degeneracy graphs: The barrier of long induced cycles. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2315–2332, 2021. doi:10.1137/1.9781611976465.138.
- 6 Andreas Bjorklund, Petteri Kaski, and Lukasz Kowalik. Counting thin subgraphs via packings faster than meet-in-the-middle time. *ACM Transactions on Algorithms*, 13(4):48:1–48:26, 2017. doi:10.1145/3125500.
- 7 Andreas Bjorklund, Rasmus Pagh, Virginia Vassilevska Williams, and Uri Zwick. Listing triangles. In *Proceedings of International Colloquium on Automata, Languages and Programming (ICALP)*, pages 223–234, 2014. doi:10.1007/978-3-662-43948-7\_19.
- 8 N. Chiba and T. Nishizeki. Arboricity and subgraph listing algorithms. *SIAM Journal of Computing*, 14(1):210–223, 1985. doi:10.1137/0214017.
- 9 Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, pages 151–158, 1971. doi:10.1145/800157.805047.
- 10 Radu Curticapean, Holger Dell, and Dániel Marx. Homomorphisms are a good basis for counting small subgraphs. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, pages 210–223, 2017. doi:10.1145/3055399.3055502.
- 11 Shiyuan Deng, Shangqi Lu, and Yufei Tao. On join sampling and the hardness of combinatorial output-sensitive join algorithms. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 99–111, 2023. doi:10.1145/3584372.3588666.
- 12 Shiyuan Deng, Francesco Silvestri, and Yufei Tao. Enumerating subgraphs of constant sizes in external memory. In *Proceedings of International Conference on Database Theory (ICDT)*, pages 4:1–4:20, 2023. doi:10.4230/LIPIcs.ICDT.2023.4.
- 13 David Eppstein. Arboricity and bipartite subgraph listing algorithms. *Information Processing Letters (IPL)*, 51(4):207–211, 1994. doi:10.1016/0020-0190(94)90121-X.
- 14 David Eppstein. Subgraph isomorphism in planar graphs and related problems. *J. Graph Algorithms Appl.*, 3(3):1–27, 1999. doi:10.7155/jgaa.00014.
- 15 David Eppstein, Maarten Löffler, and Darren Strash. Listing all maximal cliques in sparse graphs in near-optimal time. In *International Symposium on Algorithms and Computation (ISAAC)*, volume 6506, pages 403–414, 2010. doi:10.1007/978-3-642-17517-6\_36.
- 16 Peter Floderus, Mirosław Kowaluk, Andrzej Lingas, and Eva-Marta Lundell. Detecting and counting small pattern graphs. *SIAM J. Discret. Math.*, 29(3):1322–1339, 2015. doi:10.1137/140978211.
- 17 Fedor V. Fomin, Daniel Lokshtanov, Venkatesh Raman, Saket Saurabh, and B. V. Raghavendra Rao. Faster algorithms for finding and counting subgraphs. *Journal of Computer and System Sciences (JCSS)*, 78(3):698–706, 2012. doi:10.1016/j.jcss.2011.10.001.
- 18 Pierre-Louis Giscard, Nils M. Kriege, and Richard C. Wilson. A general purpose algorithm for counting simple cycles and simple paths of any length. *Algorithmica*, 81(7):2716–2737, 2019. doi:10.1007/s00453-019-00552-1.

- 19 Chinh T. Hoang, Marcin Kaminski, Joe Sawada, and R. Sritharan. Finding and listing induced paths and cycles. *Discrete Applied Mathematics*, 161(4-5):633–641, 2013. doi:10.1016/j.dam.2012.01.024.
- 20 Xiao Hu and Ke Yi. Towards a worst-case I/O-optimal algorithm for acyclic joins. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 135–150, 2016. doi:10.1145/2902251.2902292.
- 21 Xiaocheng Hu, Miao Qiao, and Yufei Tao. External memory stream sampling. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 229–239, 2015. doi:10.1145/2745754.2745757.
- 22 Xiaocheng Hu, Miao Qiao, and Yufei Tao. Join dependency testing, loomis-whitney join, and triangle enumeration. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 291–301, 2015. doi:10.1145/2745754.2745768.
- 23 Xiaocheng Hu, Miao Qiao, and Yufei Tao. I/O-efficient join dependency testing, loomis-whitney join, and triangle enumeration. *Journal of Computer and System Sciences (JCSS)*, 82(8):1300–1315, 2016. doi:10.1016/j.jcss.2016.05.005.
- 24 Manas Joglekar and Christopher Re. It’s all a matter of degree - using degree information to optimize multiway joins. *Theory Comput. Syst.*, 62(4):810–853, 2018. doi:10.1007/s00224-017-9811-8.
- 25 Bas Ketsman and Dan Suciu. A worst-case optimal multi-round algorithm for parallel computation of conjunctive queries. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 417–428, 2017. doi:10.1145/3034786.3034788.
- 26 Bas Ketsman, Dan Suciu, and Yufei Tao. A near-optimal parallel algorithm for joining binary relations. *Log. Methods Comput. Sci.*, 18(2), 2022. doi:10.46298/lmcs-18(2:6)2022.
- 27 Mahmoud Abo Khamis, Hung Q. Ngo, Christopher Ré, and Atri Rudra. Joins via geometric resolutions: Worst-case and beyond. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 213–228, 2015. doi:10.1145/2745754.2745776.
- 28 Mahmoud Abo Khamis, Hung Q. Ngo, Christopher Re, and Atri Rudra. Joins via geometric resolutions: Worst case and beyond. *ACM Transactions on Database Systems (TODS)*, 41(4):22:1–22:45, 2016. doi:10.1145/2967101.
- 29 Mahmoud Abo Khamis, Hung Q. Ngo, and Dan Suciu. What do shannon-type inequalities, submodular width, and disjunctive datalog have to do with one another? In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 429–444, 2017. doi:10.1145/3034786.3056105.
- 30 Ton Kloks, Dieter Kratsch, and Haiko Müller. Finding and counting small induced subgraphs efficiently. *Information Processing Letters (IPL)*, 74(3-4):115–121, 2000. doi:10.1016/S0020-0190(00)00047-8.
- 31 Gonzalo Navarro, Juan L. Reutter, and Javiel Rojas-Ledesma. Optimal joins using compact data structures. In *Proceedings of International Conference on Database Theory (ICDT)*, volume 155, pages 21:1–21:21, 2020. doi:10.4230/LIPIcs.ICDT.2020.21.
- 32 Jaroslav Nesetril and Svatopluk Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 26(2):415–419, 1985.
- 33 Hung Q. Ngo, Dung T. Nguyen, Christopher Re, and Atri Rudra. Beyond worst-case analysis for joins with minesweeper. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 234–245, 2014. doi:10.1145/2594538.2594547.
- 34 Hung Q. Ngo, Ely Porat, Christopher Re, and Atri Rudra. Worst-case optimal join algorithms. *Journal of the ACM (JACM)*, 65(3):16:1–16:40, 2018. doi:10.1145/3180143.
- 35 Hung Q. Ngo, Christopher Re, and Atri Rudra. Skew strikes back: new developments in the theory of join algorithms. *SIGMOD Rec.*, 42(4):5–16, 2013. doi:10.1145/2590989.2590991.
- 36 Rasmus Pagh and Francesco Silvestri. The input/output complexity of triangle enumeration. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 224–233, 2014. doi:10.1145/2594538.2594552.

- 37 Edward R. Scheinerman and Daniel H. Ullman. *Fractional Graph Theory: A Rational Approach to the Theory of Graphs*. Wiley, New York, 1997.
- 38 Yufei Tao. A simple parallel algorithm for natural joins on binary relations. In *Proceedings of International Conference on Database Theory (ICDT)*, pages 25:1–25:18, 2020. doi:10.4230/LIPIcs.ICDT.2020.25.
- 39 Todd L. Veldhuizen. Triejoin: A simple, worst-case optimal join algorithm. In *Proceedings of International Conference on Database Theory (ICDT)*, pages 96–106, 2014. doi:10.5441/002/icdt.2014.13.
- 40 Virginia Vassilevska Williams and Ryan Williams. Finding, minimizing, and counting weighted subgraphs. *SIAM Journal of Computing*, 42(3):831–854, 2013. doi:10.1137/09076619X.

## A Completing the Proof of Lemma 5

Focusing on an arbitrary hyperedge  $e = \{X_i, X_j\} \in \mathcal{E}$ , next we will explain how to process  $R_e$  in  $O((1 + \frac{D}{\sqrt{M}})^2 \cdot \frac{1}{B} + \text{sort}(N))$  I/Os such that, given any interval  $I_i \in S_{X_i}^{intv}$ , and any interval  $I_j \in S_{X_j}^{intv}$ , we can load  $R_e(I_i, I_j)$  into memory using  $O(M/B)$  I/Os.

Recall that the intervals in  $S_{X_i}^{intv}$  are disjoint. We order those intervals by their starting points; an interval has *rank*  $t$  if it has the  $t$ -th smallest starting point. Define ranks similarly for the intervals in  $S_{X_j}^{intv}$ .

For each tuple  $\mathbf{u} \in R_e$ , we associate it with an  $X_i$ -interval rank, which is the rank of the interval in  $S_{X_i}^{intv}$  covering  $\mathbf{u}(X_i)$ . By sorting, we can obtain the  $X_i$ -interval ranks of all the tuples in  $R_e$  using  $O(\text{sort}(N))$  I/Os. We also associate  $\mathbf{u}$  with an  $X_j$ -interval rank, defined similarly with respect to  $X_j$ . The  $X_j$ -interval ranks of all the tuples in  $R_e$  can again be decided using  $O(\text{sort}(N))$  I/Os.

We now sort the tuples of  $R_e$  by their  $X_i$ -interval ranks, breaking ties by  $X_j$ -interval ranks. After sorting, for any intervals  $I_i \in S_{X_i}^{intv}$  and  $I_j \in S_{X_j}^{intv}$ , the  $O(M)$  tuples in  $R_e(I_i, I_j)$  have been grouped into  $O(M/B)$  consecutive blocks on the disk. We bookmark the group's starting address such that, given the ranks of  $I_i$  and  $I_j$ , we can locate the group on the disk immediately. This requires materializing the starting addresses of all the  $|S_{X_i}^{intv}| \cdot |S_{X_j}^{intv}| = O((1 + D/\sqrt{M})^2)$  groups in a two-dimensional array, which can be created in  $O(1 + (1 + \frac{D}{\sqrt{M}})^2 \cdot \frac{1}{B} + \frac{N}{B})$  I/Os by scanning the (sorted)  $R_e$  once.

## B Preprocessing the Input Relations for Proving Lemma 6

We will first explain how to preprocess  $R$  in  $O((\frac{|R|}{\lambda})^2 \cdot \frac{1}{B} + \text{sort}(|R|))$  I/Os such that, for any interval  $I_X \in S_X^{intv}$  and any interval  $I_Y \in S_Y^{intv}$ , we can ensure that (i)  $R(I_X, I_Y)$  is stored in consecutive blocks and (ii) the first block's address can be obtained in one I/O.

It suffices to re-apply the techniques illustrated in Appendix A. Sort the intervals of  $S_X^{intv}$  by their starting points and do the same for  $S_Y^{intv}$ , which requires  $O(\text{sort}(|R|/\lambda))$  I/Os. An interval of  $S_X^{intv}$  (resp.,  $S_Y^{intv}$ ) is said to have *rank*  $t$  if it has the  $t$ -th smallest starting point among all the intervals in  $S_X^{intv}$  (resp.,  $S_Y^{intv}$ ). Associate each tuple  $\mathbf{u} \in R$  with an  $X$ - (resp.,  $Y$ -) *interval rank*, which is the rank of the interval in  $S_X^{intv}$  covering  $\mathbf{u}(X)$  (resp.,  $S_Y^{intv}$  covering  $\mathbf{u}(Y)$ ). Sort the tuples of  $R$  by their  $X$ -interval ranks, breaking ties by their  $Y$ -interval ranks, after which the tuples in  $R(I_X, I_Y)$  are placed in a sequence of consecutive blocks for any interval pair  $(I_X, I_Y) \in S_X^{intv} \times S_Y^{intv}$ . We record the sequence's starting address such that the sequence can be located immediately based on the ranks of  $I_X$  and  $I_Y$ . This requires materializing  $|S_X^{intv}| \cdot |S_Y^{intv}| = O((|R|/\lambda)^2)$  starting addresses in a two-dimensional array, which can be created in  $O(1 + (\frac{|R|}{\lambda})^2 \cdot \frac{1}{B} + \frac{|R|}{B})$  I/Os by scanning  $R$  once.

We now turn our attention to preprocessing  $S_1, S_2, \dots, S_\ell$ . For each  $i \in [\ell]$ , recall that  $S_i$  has been sorted on attribute  $X$ . By scanning  $S_i$  synchronously with the (already sorted)  $S_X^{intv}$ , we can ensure that, for each interval  $I_X \in S_X^{intv}$ , the set  $S_i(I_X)$  is stored in a sequence of consecutive blocks. We record the sequence's starting address such that the sequence can be located directly based on the rank of  $I_X$ . This requires materializing  $|S_X^{intv}| = O(\lceil |R|/\lambda \rceil)$  starting addresses in an array, which can be created in  $O(1 + \frac{|R|}{\lambda B} + \frac{|S_i|}{B})$  I/Os by scanning  $S_i$  once. The total I/O cost of preprocessing the  $S_i$  of all  $i \in [\ell]$  this way is  $O(\ell + \ell \frac{|R|}{\lambda B} + \sum_{i=1}^{\ell} |S_i|/B)$ .

The preprocessing of  $T_1, T_2, \dots, T_\ell$  is similar and omitted.

## C Proof of Equation (14)

We will prove the following relationship for each  $i \in [\ell]$ :

$$\begin{aligned} & \sum_{(I_X, I_Y) \in S_X^{intv} \times S_Y^{intv}} \left\lceil \frac{|R(I_X, I_Y)|}{M} \right\rceil \cdot \frac{|S_i(I_X)| + |T_i(I_Y)| + M}{B} \\ &= O\left(\left(\frac{|R|}{\lambda} + \frac{\lambda}{M}\right) \frac{|S_i| + |T_i|}{B} + \frac{|R|}{B} + \frac{|R|^2 M}{\lambda^2 B}\right). \end{aligned} \quad (25)$$

Equation (14) will then follow from a simple summation on all  $i \in [\ell]$ .

Clearly:

$$\begin{aligned} & \text{left hand side of (25)} \\ & \leq \sum_{(I_X, I_Y)} \left(1 + \frac{|R(I_X, I_Y)|}{M}\right) \cdot \frac{|S_i(I_X)| + |T_i(I_Y)| + M}{B} \\ &= \sum_{(I_X, I_Y)} \frac{|S_i(I_X)| + |T_i(I_Y)| + M}{B} + \sum_{(I_X, I_Y)} \frac{|R(I_X, I_Y)|}{M} \cdot \frac{|S_i(I_X)| + |T_i(I_Y)| + M}{B}. \end{aligned} \quad (26)$$

The first summation of (26) is easy to bound:

$$\begin{aligned} \sum_{(I_X, I_Y)} \frac{|S_i(I_X)| + |T_i(I_Y)| + M}{B} &= \sum_{(I_X, I_Y)} \frac{|S_i(I_X)|}{B} + \sum_{(I_X, I_Y)} \frac{|T_i(I_Y)|}{B} + \sum_{(I_X, I_Y)} \frac{M}{B} \\ &= \frac{|T_Y^{intv}| \cdot |S_i|}{B} + \frac{|S_X^{intv}| \cdot |T_i|}{B} + \frac{M}{B} \cdot |S_X^{intv}| \cdot |T_Y^{intv}| \\ &= O\left(\frac{|R|(|S_i| + |T_i|)}{\lambda \cdot B} + \frac{M |R|^2}{B \lambda^2}\right) \end{aligned} \quad (27)$$

where the last step used  $|S_X^{intv}| = O(|R|/\lambda)$  and  $|T_Y^{intv}| = O(|R|/\lambda)$ . To unfold the second summation of (26), let us first rearrange the terms:

$$\begin{aligned} & \sum_{(I_X, I_Y)} \frac{|R(I_X, I_Y)|}{M} \cdot \frac{|S_i(I_X)| + |T_i(I_Y)| + M}{B} \\ &= \sum_{(I_X, I_Y)} \frac{|R(I_X, I_Y)|}{M} \frac{|S_i(I_X)|}{B} + \sum_{(I_X, I_Y)} \frac{|R(I_X, I_Y)|}{M} \frac{|T_i(I_Y)|}{B} + \sum_{(I_X, I_Y)} \frac{|R(I_X, I_Y)|}{B} \end{aligned} \quad (28)$$

To facilitate the subsequent derivation, let us define

$$R(I_X, -) = \{\mathbf{u} \in R \mid \mathbf{u}(X) \in I_X\}$$

## 21:20 Subgraph Enumeration in Optimal I/O Complexity

for any interval  $I_X \in S_X^{intv}$ . Note that the size of  $R(I_X, -)$  is bounded by  $O(\lambda)$  due to the way that  $S_X^{intv}$  is constructed. Equipped with this, we can derive

$$\begin{aligned}
 \sum_{(I_X, I_Y)} |R(I_X, I_Y)| |S_i(I_X)| &= \sum_{I_X} |S_i(I_X)| \cdot \left( \sum_{I_Y} |R(I_X, I_Y)| \right) \\
 &= \sum_{I_X} |S_i(I_X)| \cdot |R(I_X, -)| \\
 &= O(\lambda) \cdot \sum_{I_X} |S_i(I_X)| = O(\lambda \cdot |S_i|). \tag{29}
 \end{aligned}$$

A symmetric analysis shows that  $\sum_{(I_X, I_Y)} |R(I_X, I_Y)| |T_i(I_Y)| = O(\lambda \cdot |T_i|)$ . Utilizing also the obvious fact  $\sum_{(I_X, I_Y)} |R(I_X, I_Y)| = |R|$ , we obtain

$$(28) = O\left(\frac{\lambda \cdot |S_i|}{MB} + \frac{\lambda \cdot |T_i|}{MB} + \frac{|R|}{B}\right). \tag{30}$$

Equation (25) now follows from (26), (27) and (30).

# Evaluating Graph Queries Using Semantic Treewidth

**Cristina Feier**

University of Warsaw, Poland

**Tomasz Gogacz**

University of Warsaw, Poland

**Filip Murlak**

University of Warsaw, Poland

---

## Abstract

Unions of conjunctive two-way regular path queries (UC2RPQs) are a common abstraction of query languages for graph databases, much like unions of conjunctive queries (UCQs) in the relational case. As in the case of UCQs, their evaluation is NP-complete in combined complexity. Semantic tree-width, i.e. the minimal treewidth of equivalent queries, has been proposed as a candidate criterion to characterize fixed-parameter tractability of UC2RPQs. It was recently shown how to decide the semantic tree-width of a UC2RPQ, by constructing the best under-approximation of a given treewidth, in the form of a UC2RPQ of size doubly exponential in the size of the original query. This leads to an fpt algorithm for evaluating UC2RPQs of semantic TW  $k$  which runs in time doubly exponential in the size of the parameter, i.e. in the UC2RPQ. Here we describe a more efficient fpt algorithm for evaluating UC2RPQs of semantic treewidth  $k$  which runs in time singly exponential in the size of the parameter. We do this by a careful construction of a witness query which, while still being doubly exponential, can be represented as a Datalog program of bounded width and singly exponential size.

**2012 ACM Subject Classification** Theory of computation → Regular languages; Information systems → Query languages; Theory of computation → Semantics and reasoning; Theory of computation → Automated reasoning; Theory of computation → Complexity theory and logic

**Keywords and phrases** conjunctive two-way regular path queries, fixed-parameter tractable evaluation, semantic treewidth, Datalog encoding, optimization

**Digital Object Identifier** 10.4230/LIPIcs.ICDT.2024.22

**Funding** This work was supported by Poland’s National Science Centre grant 2018/30/E/ST6/00042.

## 1 Introduction

“The future is big graphs” [21]. Already today graph databases are mainstream in a wide range of application domains, including social networks, fraud detection, biological networks, bioinformatics, cheminformatics, medical data, and knowledge management [3]. By 2025, as Gartner predicts, graph technologies will be used in 80% of data and analytics innovations. The landscape of graph technologies, however, is currently very fragmented, with multiple vendors offering their own query languages (e.g. [13, 23]). This might change with the upcoming Graph Query Language Standard [11], but for now theoretical research naturally focuses on abstract formalisms, capturing the core of the multiple query languages.

While for relational databases conjunctive queries (CQs) and their unions (UCQs) are a premier abstract query language, graph databases are typically queried using *conjunctive regular path queries* (CRPQs) and unions thereof (UCRPQs) which generalize UCQs by replacing atoms with *regular path queries* (RPQs) [19]. RPQs specify connections between graph nodes using regular expressions over edge labels. If edges can be traversed both ways, one speaks of *two-way regular path queries* (2RPQs) and (*unions of*) *conjunctive two-way*



© Cristina Feier, Tomasz Gogacz, and Filip Murlak;  
licensed under Creative Commons License CC-BY 4.0  
27th International Conference on Database Theory (ICDT 2024).

Editors: Graham Cormode and Michael Shekelyan; Article No. 22; pp. 22:1–22:20

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



*regular path queries*, abbreviated as (U)C2RPQs. While the original semantics for such queries was based on the existence of simple paths, the most commonly adopted semantics nowadays relies on unrestricted paths. Under the latter semantics, 2RPQs can be evaluated in polynomial time, but the complexity of evaluating C2RPQs is the same as for CQs: NP-complete in combined complexity.

In the classical setting, the intractability of CQs lead to a long line of research, where different structural measures on queries were proposed in an attempt to identify cases where evaluation can be done efficiently. This spans from the well-known polynomial time algorithm of Yannakakis for evaluating acyclic CQs [24] to tractability results for queries of bounded treewidth [9], bounded (fractional) hypertreewidth [14, 16], and culminates with the result of Grohe [15] that establishes a dichotomy for fixed-parameter tractable evaluation of CQs in the bounded arity setting.

Under an assumption from parameterized complexity theory, that  $W[1] \neq FPT$ , Grohe's result establishes that exactly those classes of UCQs which have bounded treewidth modulo equivalence (also known as *bounded semantic treewidth*) can be evaluated in FPT. Furthermore, bounded semantic treewidth also guarantees tractability in a classical sense, i.e. polynomial-time combined complexity. In the unbounded arity setting, a similar dichotomy (again subject to an assumption, the Exponential Time Hypothesis [17]) has been established, this time using another measure called submodular width [18, 10].

In the same vein, Barcelo et al. [4, 5] initiated the investigation of efficient evaluation of UC2RPQs. They showed that acyclic UC2RPQs can be evaluated in polynomial time (even linear in particular cases). Semantically acyclic UC2RPQs [6] enjoy the same good properties, although deciding whether a UC2RPQ is semantically acyclic is EXPSPACE-complete (unlike for UCQs, for which it is NP-complete). This is not surprising given that the containment problem for UC2RPQs is EXPSPACE-complete [8].

As concerns semantic treewidth of (U)C2RPQs, Romero et al. [20] introduced two notions of C2RPQ equivalence: one based on homomorphisms, and another based on logical equivalence. They show that there exists a notion of homomorphism for C2RPQs under which bounded treewidth modulo equivalence guarantees polynomial evaluation. For logical equivalence, however, tractability is no longer achievable, so the focus shifts to fixed parameter tractability. In this case, it follows from existing results on UCQ evaluation and RPQ evaluation that UC2RPQs of bounded treewidth are fixed-parameter tractable: one can simply materialize all the RPQs in the database and then regard the C2RPQ as a CQ over the materialized database. The authors lift this to UC2RPQs of bounded semantic treewidth, i.e. UC2RPQs that are logically equivalent to one of bounded treewidth. They achieve this by computing a so-called witness for bounded semantic treewidth, whose actual treewidth might be up to  $2k + 1$ , when the semantic treewidth of the query is  $k$ . Again, according to results on UCQ evaluation, together with the complexity of evaluating RPQs, such a query can be evaluated in time  $O(f(\Phi)|\mathcal{G}|^{2k+1})$ , where  $|\Phi|$  is the size of the query,  $|\mathcal{G}|$  is the size of the graph database and  $f$  is a singly exponential function.

It remained open how to decide semantic TW of UC2RPQs, and, in particular, how to construct a witness of optimal TW. This has been settled recently by Figueira and Morvan [12], who construct for a given  $k$  the best (in the sense of tightest) under-approximations of treewidth  $k$  of the original query. Such under-approximations have in the worst case size doubly exponential in the size of the original query. As such an under-approximation is equivalent to the original query when the semantic treewidth of the query is  $k$ , this leads to an algorithm for evaluating UC2RPQs of semantic TW  $k$  which runs in time  $O(f(\Phi)|\mathcal{G}|^{k+1})$ , with  $f$  being a doubly exponential function. While this is a significant improvement in data

complexity, as it lowers the exponent of the size of data from  $2k + 1$  to  $k + 1$  compared to existing algorithms, the algorithm is impractical due to its doubly exponential combined complexity.

In this paper, we zoom into the problem of evaluating a UC2RPQ  $\Phi$  of semantic TW  $k$  and show that it is possible to evaluate  $\Phi$  in time  $O(g(\Phi)|\mathcal{G}|^{k+1})$ , with  $g$  a singly exponential function. We do this by a careful construction of optimal approximations of a given treewidth (and implicitly of semantic treewidth witness queries). We construct several such approximations, starting with an infinitary one which we refine to ensure some good structural properties, and ending with a finite witness of bounded size. As the finite witness in the worst case is still doubly exponential in the size of  $\Phi$  (as in [12]) we cannot use it to get better complexity bounds. However, we can exploit the infinitary witness with good structural properties. This query can be compiled into a union of singly exponentially many queries, called *skeleton* queries.

Skeleton queries group together C2RPQs which share some common structure, the *skeleton*. They can be seen as tree-shaped conjunctions of so-called *type reachability queries*. Such type reachability queries encode all ways in which a top  $k + 1$  tuple can be reached from a bottom  $k + 1$  tuple via a path query of width  $k$ . While there might be doubly exponentially many such ways to reach a tuple, the type reachability query can be encoded via an exponential sized Datalog program of width  $k + 1$  by employing suitable notions of types and compatibility between such types. As concerns skeleton queries, their tree-based structure makes them amenable to evaluation via a dynamic programming approach. The approach is reminiscent of Yannakakis' algorithm for evaluating acyclic CQs [24], if one regards type reachability queries as oracles that compute a set of target  $(k + 1)$ -tuples when seeded with a set of source  $(k + 1)$ -tuples. Again, this step can be encoded via a Datalog program of width  $k + 1$  and polynomial size. By executing such a Datalog program, we obtain an evaluation procedure which runs within the desired time bounds.

The paper is organized as follows. We start with some preliminaries in Section 2. Section 3 is dedicated to constructing TW  $k$  approximations. Then, in Section 4 we encode the evaluation of TW  $k$  approximations into a Datalog program, to achieve higher efficiency. Finally, in Section 5 we conclude and discuss future work.

## 2 Preliminaries

### Relational structures, graph databases

A *schema*  $\mathbf{S}$  is a finite set of relational symbols with associated arities. An  *$\mathbf{S}$ -fact* has the form  $r(\mathbf{a})$ , where  $r \in \mathbf{S}$ , and  $\mathbf{a}$  is a tuple of constants of size the arity of  $r$ . An  *$\mathbf{S}$ -structure*  $A$  is a set of  $\mathbf{S}$ -facts. The domain of a structure  $A$ ,  $\text{dom}(A)$ , is the set of constants which occur in facts in  $A$ . A structure  $A$  *maps into* a structure  $B$ , written  $A \rightarrow B$  if there exists a function  $h : \text{dom}(A) \rightarrow \text{dom}(B)$  such that for every fact  $r(\mathbf{a})$  in  $A$ , there exists a fact  $r(h(\mathbf{a}))$  in  $B$ , where  $h(\mathbf{a})$  is the tuple obtained from  $\mathbf{a}$  by pointwise application of  $h$ . In this case,  $h$  is said to be a *homomorphism from  $A$  to  $B$* . The Gaifman graph of a structure  $A$  is an undirected graph  $(V, E)$  with  $V = \text{dom}(A)$  and  $\{a_1, a_2\} \in E$  iff there exists some fact  $r(\mathbf{a})$  in  $A$  such that both  $a_1$  and  $a_2$  occur in  $\mathbf{a}$ .

In the following, let  $\Sigma$  be some countable alphabet. A *graph database*  $\mathcal{G}$  over  $\Sigma$  is a finite directed graph with edges labeled with symbols from  $\Sigma$ . It can be seen as a relational structure over the schema  $\{R_a \mid a \in \Sigma\}$ , where each relational symbol is binary. A path  $p$  in a graph database is a sequence  $(u_1 \xrightarrow{a_1} u_2 \dots u_l \xrightarrow{a_l} u_{l+1})$  denoting that for every  $1 \leq i \leq l$ ,  $R_{a_i}(u_i, u_{i+1})$  is in  $\mathcal{G}$ . The label of  $p$  as above, denoted  $\lambda(p)$ , is the word  $a_1 \dots a_l$  from  $\Sigma^*$ .

## Regular expressions, automata

A *non-deterministic finite automata (NFA)*  $A$  is a tuple  $(Q, \Sigma, \delta, s_0, F)$ , where  $Q$  is the *set of states*,  $\delta: Q \times \Sigma \rightarrow 2^Q$  is the *transition function*,  $s_0$  is the *initial state* and  $F \subseteq Q$  is a *set of final states*. To access the set of states of any given automaton we use the function  $\text{states}(\cdot)$ ; that is,  $\text{states}(A) = Q$ . A *run of  $A$*  on a word  $w \in \Sigma^*$  is a sequence of states of the form  $(s_0, \dots, s_n)$ , where  $n = |w|$  and for every  $1 \leq i \leq n$ ,  $s_i \in \delta(w_i, s_{i-1})$ . Automaton  $A$  accepts a word  $w$  if there exists a run  $(s_0, \dots, s_n)$  of  $A$  on  $w$  such that  $s_n \in F$ .

Every regular expression can be represented via an NFA of linear size. For such an expression  $L$ , we denote with  $A(L)$  a fixed linear-size NFA for  $L$ . Conversely, each NFA  $A$  can be converted into a regular expression  $E(A)$ . For an NFA  $A$  and two states  $s, s'$  we denote with  $A(s, s')$  the NFA having the same set of states and transition function as  $A$ , but having  $s$  as initial state and  $s'$  as a unique final state. When convenient, we abbreviate  $E(A(L)(s, s'))$  as  $L[s, s']$ . While  $L[s, s']$  may be exponentially larger than  $L$  as its computation involves the determinization of the NFA  $L[22]$ , we actually always work with the NFA representation, whose size does not grow.

## UC2RPQs

A *regular path query (RPQ)* is a query of the form  $L(u, v)$  where  $L$  is a regular expression over  $\Sigma$ . For a graph database  $\mathcal{G}$  and  $u', v' \in \text{dom}(\mathcal{G})$ , it is the case that  $\mathcal{G} \models L(u', v')$  if there exists a path  $p$  in  $\mathcal{G}$  from  $u'$  to  $v'$  such that  $\lambda(p) \in L$ . When one is interested in two-way navigation along labeled edges in a graph database, the regular expression  $L$  can be over the alphabet  $\Sigma^\pm = \Sigma \cup \{a^- \mid a \in \Sigma\}$ . In this case,  $L(u, v)$  is said to be a *two-way RPQ (2RPQ)* and it is evaluated over the *completion*  $\mathcal{G}^\pm$  of a graph database with “reverse” edges:  $\mathcal{G}^\pm = \mathcal{G} \cup \{R_{a^-}(v, u) \mid R_a(u, v) \in \mathcal{G}\}$ . A *conjunctive two-way regular path query (C2RPQ)*  $\phi(\mathbf{v}')$  is a query of the form  $\exists \mathbf{u}' \bigwedge_{1 \leq i \leq n} L(u_i, v_i)$  where each  $L(u_i, v_i)$  is a 2RPQ and  $(\mathbf{v}', \mathbf{u}')$  is a disjoint partition of the set of variables  $\mathbf{v} \cup \mathbf{u}$  occurring in the query. The variables  $\mathbf{v}'$  are called the *free variables* of  $\phi$ . When  $\mathbf{v}' = \emptyset$ , we say that  $\phi$  is *Boolean*. If  $\mathbf{v}' = \mathbf{v} \cup \mathbf{u}$ ,  $\phi(\mathbf{v}')$  is said to be a *full C2RPQ*. If all atoms in a C2RPQ  $\phi$  are of the form  $L(u, v)$ , with  $L$  a language over  $\Sigma$ , we say that  $\phi$  is a *conjunctive query (CQ)*. For  $\mathcal{G}$  a graph database and  $\mathbf{a}$  a tuple of constants from  $\text{dom}(\mathcal{G})$ , it is the case that  $\mathcal{G} \models \phi(\mathbf{a})$  if there exists some mapping  $h: \mathbf{u} \cup \mathbf{v} \rightarrow \text{dom}(\mathcal{G})$  such that  $h(\mathbf{v}') = \mathbf{a}$  and  $\mathcal{G} \models L(h(u_i), h(v_i))$ , for every  $1 \leq i \leq n$ . When  $\phi$  is Boolean, we say that  $h$  is a *homomorphism* from  $\phi$  to  $\mathcal{G}$ .

A UC2RPQ  $\Phi(\mathbf{v}')$  is a union of C2RPQs with free variables  $\mathbf{v}'$ . When  $\mathbf{v}' = \emptyset$ ,  $\Phi$  is said to be *Boolean*. For  $\mathcal{G}$  a graph database and  $\mathbf{a}$  a tuple of constants from  $\text{dom}(\mathcal{G})$ ,  $\mathcal{G} \models \Phi(\mathbf{a})$ , if  $\mathcal{G} \models \phi(\mathbf{a})$  for some C2RPQ  $\phi$  in  $\Phi$ . We say that a UC2RPQ  $\Phi_1$  is *contained* in another UC2RPQ  $\Phi_2$ , denoted  $\Phi_1 \subseteq \Phi_2$ , if for every graph database  $\mathcal{G}$  and every tuple  $\mathbf{a}$  of constants from  $\text{dom}(\mathcal{G})$ ,  $\mathcal{G} \models \Phi_1(\mathbf{a})$  implies  $\mathcal{G} \models \Phi_2(\mathbf{a})$ . If  $\Phi_1 \subseteq \Phi_2$  and  $\Phi_2 \subseteq \Phi_1$ , we say that  $\Phi_1$  and  $\Phi_2$  are *equivalent*, and we write  $\Phi_1 \equiv \Phi_2$ .

## Trees, tree decompositions, semantic treewidth

A *tree*  $T$  is represented as an acyclic undirected graph  $(V, E)$ , with a distinguished node  $r \in V$ , the *root*. Given a tree  $T$  as above and two nodes  $v_1, v_2 \in V$ , the unique simple path in  $T$  from  $v_1$  to  $v_2$  is denoted as  $\text{path}_T(v_1, v_2)$ . We write  $v \leq_T v'$  if  $v \in \text{path}_T(r, v')$  and  $v <_T v'$  if  $v \in \text{path}_T(r, v')$  and  $v \neq v'$ . The *lowest common ancestor of  $v_1$  and  $v_2$  in  $T$* , denoted as  $\text{lca}_T(v_1, v_2)$ , is the node  $v$  such that  $v \leq_T v_1$ ,  $v \leq_T v_2$ , and  $v \in \text{path}_T(v_1, v_2)$ . We write  $v_1 \not\leq_T v_2$  if  $v_1$  and  $v_2$  are  *$\leq_T$ -incomparable*; that is, neither  $v_1 \leq_T v_2$  nor  $v_1 \geq_T v_2$ .

A *tree decomposition* of a graph  $G = (V, E)$  is a pair  $\delta = (T_\delta, \chi)$ , with  $T_\delta = (V_\delta, E_\delta)$  a tree with root  $r_\delta$ , and  $\chi$  a labeling function  $V_\delta \rightarrow 2^V$  such that:

1.  $\bigcup_{t \in V_\delta} \chi(t) = V$ .
2. If  $(v_1, v_2) \in E$ , then  $v_1, v_2 \subseteq \chi(t)$  for some  $t \in V_\delta$ .
3. For each  $v \in V$ , the set of nodes  $\{t \in V_\delta \mid v \in \chi(t)\}$  induces a connected subtree of  $T_\delta$ .

We will refer to elements of  $V_\delta$  as *bags* of the tree decomposition and to  $\chi(v)$  as the *content* of bag  $v$ , for  $v \in V_\delta$ . For each  $u \in V$ , we denote with  $u_\delta$  the root of the subtree  $\{v \in V_\delta \mid u \in \chi(v)\}$  of  $T_\delta$ , i.e. the node  $v$  from  $V_\delta$  for which there is no other node  $v' \in V_\delta$  such that  $u \in \chi(v')$ , and  $v' <_{T_\delta} v$ . The width of  $\delta$  is  $\max_{v \in V_\delta} (|\chi(v)| - 1)$ .

The *treewidth* ( $TW$ ) of  $G$ ,  $TW(G)$ , is the smallest  $k$  such that there exists a tree decomposition  $\delta$  of  $G$  of width  $k$ .

For a graph database  $D$ , the treewidth of  $D$ ,  $TW(D)$ , is the treewidth of its Gaifman graph  $G_D$ . For a Boolean C2RPQ  $\phi$  with variables  $\mathbf{v}$ , we denote with  $G_\phi$ , the graph  $(V, E)$  with  $V = \mathbf{v}$  and  $E = \{(u, v) \mid L(u, v) \in \phi\}$ . The treewidth of  $\phi$ , denoted  $TW(\phi)$ , is  $TW(G_\phi)$ . The treewidth of a Boolean UC2RPQ  $\Phi$  is the maximum treewidth among its C2RPQs:  $TW(\Phi) = \max_{\phi \in \Phi} TW(\phi)$ .

The *semantic treewidth* of a UC2RPQ  $\Phi$  is the minimum treewidth of a UC2RPQ which is equivalent to it.

## Parameterized complexity

For a finite alphabet  $\Sigma$ , a *parameterized problem* is a tuple  $(P, \kappa)$ , where  $P \subseteq \Sigma^*$  is a problem, and  $\kappa: \Sigma^* \rightarrow \mathbb{N}$  is a PTIME computable function called the *parameterization* of  $P$ . Such a parameterized problem is *fixed-parameter tractable* if there exists an algorithm for deciding  $P$  for an input  $x \in \Sigma^*$  in time  $f(\kappa(x))poly(|x|)$ , where  $f$  is a computable function and *poly* is a polynomial. The class of all fixed-parameter tractable problems is denoted as FPT. In this paper we are interested in the parameterized problem of evaluating Boolean UC2RPQs, where the parameter is the size of the query.

## Datalog

A *term* is a constant or variable. An *atom* is of the form  $p(t_1, \dots, t_n)$ , where  $p$  is a predicate symbol of arity  $n$  and  $t_1, \dots, t_n$  are terms. An atom is said to be *ground* if it contains no variables. A Datalog program  $\Pi$  is a set of rules of the form

$$(r): \beta(\mathbf{x}, \mathbf{y}) \rightarrow a(\mathbf{x}),$$

where  $\beta(\mathbf{x}, \mathbf{y})$  is a set of atoms having as terms constants or variables from  $\mathbf{x} \cup \mathbf{y}$  (called the *body* of  $r$ ), and  $a(\mathbf{x})$  is an atom with variables  $\mathbf{x}$  (called the *head* of  $r$ ). Predicate symbols which occur only in atoms in the body of rules are called *EDBs*, while all other predicates are called *IDBs*. The maximum number of variables occurring in some rule in  $\Pi$  is the *width* of  $\Pi$ , denoted  $w(\Pi)$ .

A structure  $I$  satisfies a Datalog rule  $r$  as above, if for every function  $h: \mathbf{x} \cup \mathbf{y} \rightarrow \text{dom}(I)$  which is a homomorphism from  $\beta(\mathbf{x}, \mathbf{y})$  to  $I$ ,  $a(h(\mathbf{x})) \in I$ . Given a database  $D$  and a Datalog program  $\Pi$ , a structure  $I$  is a *model* of  $\Pi$  if it extends  $D$  (we adopt the *standard name assumption* – constants in  $\text{dom}(D)$  are interpreted as themselves) and satisfies every rule from  $\Pi$ . For a ground atom  $a$ , we say that  $(\Pi, D) \models a$  if for every model  $I$  of  $\Pi$  and  $D$ ,  $I \models a$ .

Datalog programs (in conjunction with DBs) have the property that they have a finite minimal model [1], i.e. a model  $M_{\Pi,D}$  which is a sub-structure of every other model of  $\Pi$  and  $D$ . The minimal model  $M_{\Pi,D}$  can be computed via a fix-point procedure (called *naive evaluation* or *chase*) which constructs a sequence of converging instances  $(I_i)_{i \geq 0}$  as follows:  $I_0 = D$ ; given  $I_i$ ,  $I_{i+1}$  is obtained by considering for every rule in  $\Pi$  all homomorphisms  $h$  from  $\beta(\mathbf{x}, \mathbf{y})$  to  $I_i$  and adding  $a(\mathbf{x})$  to  $I_i$ . Then,  $M_{\Pi,D} = \bigcup I_i$ . The procedure runs in time  $O(f(|\Pi|)|D|^{w(\Pi)})$ , where  $f$  is a polynomial function.

### 3 Treewidth- $k$ Approximations

In this section we investigate how one can approximate a given UC2RPQ with a UC2RPQ of a given treewidth.

► **Definition 1.** A UC2RPQ  $\Phi'$  is a TW- $k$  approximation of a UC2RPQ  $\Phi$  if

- (i)  $\Phi'$  has TW  $k$ ;
- (ii)  $\Phi' \subseteq \Phi$ ; and
- (iii) there is no UC2RPQ  $\Phi''$  of TW  $k$  such that  $\Phi' \subset \Phi'' \subseteq \Phi$ .

The actual goal of this section is to develop a toolbox for handling TW- $k$  approximations, suitable for our approach to query evaluation, described in Section 4. However, to focus our attention, we set a local goal of re-proving the following result, established in [12].

► **Theorem 2.** For every UC2RPQ  $\Phi$  and  $k > 1$ , it is possible to construct a TW- $k$  approximation  $\Phi_k$  of size doubly exponential in the size of  $\Phi$ .

Note that once we know how to construct TW- $k$  approximations, we can compute the semantic treewidth of a given UC2RPQ by constructing TW- $k$  approximations for increasing values of  $k$  and testing their equivalence to the original UC2RPQ. It is known that checking containment of UC2RPQs is decidable and can actually be performed in EXPSPACE [8].

We construct the TW- $k$  approximation in several steps, starting from an infinitary UC2RPQ and finally obtaining a UC2RPQ with the desired size bounds. For the initial step, we use the following sufficient condition.

► **Lemma 3.** Let  $\Phi$  be a UC2RPQ and  $\Phi'$  be a UC2RPQ of TW  $k$  which is contained in  $\Phi$  and is equivalent to  $\Phi$  on graph databases of TW  $k$ . Then  $\Phi'$  is a TW- $k$  approximation of  $\Phi$ .

The proof of Lemma 3 uses the notion of *expansion*, which is a graph database of a certain shape which satisfies a C2RPQ.

► **Definition 4.** Given a C2RPQ  $\Phi$  and a graph database  $\mathcal{G}$  such that  $\mathcal{G} \models \Phi$ , we say that  $\mathcal{G}$  is an expansion of  $\Phi$  if there exists an injective homomorphism  $h$  from  $\Phi$  to  $\mathcal{G}$  such that  $\mathcal{G}$  can be seen as a set of paths of the form  $h(u_i) \rightarrow u'_2 \rightarrow u'_3 \rightarrow \dots \rightarrow h(v_i)$ , one for each atom of the form  $L(u_i, v_i)$  in  $\Phi$ , that are pairwise disjoint except (possibly) for their endpoints.

The following lemma summarises some properties of expansions:

► **Lemma 5 (Folklore).** Let  $\Phi$  be a C2RPQ. The following hold:

1. if  $\Phi$  has TW  $k$ , with  $k > 1$ , so does every expansion  $\mathcal{G}$  of  $\Phi$
2. for every graph database  $\mathcal{G}$  such that  $\mathcal{G} \models \Phi$ , there exists an expansion  $\mathcal{G}'$  of  $\Phi$  such that  $\mathcal{G}' \rightarrow \mathcal{G}$ .

We are ready to provide the proof of Lemma 3.

**Proof.** Towards contradiction, suppose that  $\Phi'$  is not a TW- $k$  approximation of  $\Phi$  and let  $\Phi''$  be one. Then, there exists a graph database  $\mathcal{G}$  such that  $\mathcal{G} \models \Phi''$  but  $\mathcal{G} \not\models \Phi'$ . From Lemma 5, there must be an expansion  $\mathcal{G}'$  of  $\Phi''$  of TW  $k$  such that  $\mathcal{G}' \rightarrow \mathcal{G}$ . Clearly,  $\mathcal{G}' \models \Phi''$ . Since  $\Phi'' \subseteq \Phi$ ,  $\mathcal{G}' \models \Phi$ . As  $\mathcal{G}'$  has TW  $k$  and  $\Phi' \equiv_k \Phi$ ,  $\mathcal{G}' \models \Phi'$ . It follows immediately that  $\mathcal{G} \models \Phi'$ , which contradicts the initial assumption and concludes the proof.  $\blacktriangleleft$

In the following, we will denote with  $\equiv_k$  the notion of equivalence of UC2RPQs on TW- $k$  databases. Similarly, for  $\subseteq_k$ .

For technical convenience, we shall assume that each C2RPQ has at most one atom of the form  $L(x, y)$  for each regular expression  $L$ . This is without loss of generality, as we can always replace different occurrences of the same regular expression by equivalent but syntactically different regular expressions. By a *path* in a C2RPQ  $\psi$  we mean a sequence  $A_0(x_0, x_1), A_1(x_1, x_2), \dots, A_n(x_n, x_{n+1})$  of atoms from  $\psi$ . We say the path is *simple* if variables  $x_0, x_1, \dots, x_{n+1}$  are all different.

Following [20], we rely on *subdivisions* and *quotients* of C2RPQs, which we recall below. For a UC2RPQ  $\Phi$ , and some  $r \geq 1$ , the set  $\mathbb{SD}_r(\Phi)$  of  $r$ -*subdivisions* of  $\Phi$  is the set of all C2RPQs that can be obtained from a C2RPQ  $\phi$  in  $\Phi$  as follows: for each atom  $L(u, v)$  in  $\phi$ , consider fresh variables  $u_1, \dots, u_l$ , with  $l < r$ , and a sequence of states  $s_0, s_1, \dots, s_{l+1}$  of  $A(L)$  such that  $s_0$  and  $s_{l+1}$  are an initial and a final state of  $A(L)$ , and replace  $L(u, v)$  in  $\phi$  with a path  $L[s_0, s_1](u, u_1), L[s_1, s_2](u_1, u_2), \dots, L[s_l, s_{l+1}](u_l, v)$ . We will refer to this path as the  $L$ -*path* and to the introduced atoms as the  $L$ -*atoms*. For a set  $S$  of C2RPQs, we write  $\mathbb{Q}(S)$  for the set of *quotients* of C2RPQ in  $S$ ; that is, C2RPQs that can be obtained from a C2RPQ in  $S$  by variable identification. We let  $\mathbb{Q}_k(S)$  be the set of C2RPQs from  $\mathbb{Q}(S)$  which have treewidth  $k$ .

With that, we can make our first step in constructing the desired approximation of  $\Phi$ . Let  $\Phi_{k, \infty}$  be the infinitary UC2RPQ consisting of all C2RPQs from  $\bigcup_{r>1} \mathbb{Q}_k(\mathbb{SD}_r(\Phi))$ .

► **Lemma 6.** *For each UC2RPQ  $\Phi$ ,  $\Phi_{k, \infty}$  is a TW- $k$  approximation of  $\Phi$ .*

The next step is to simplify the structure of C2RPQs building up the TW- $k$  approximation. This will help us ensure the desired size bounds, and will be crucial in the next section.

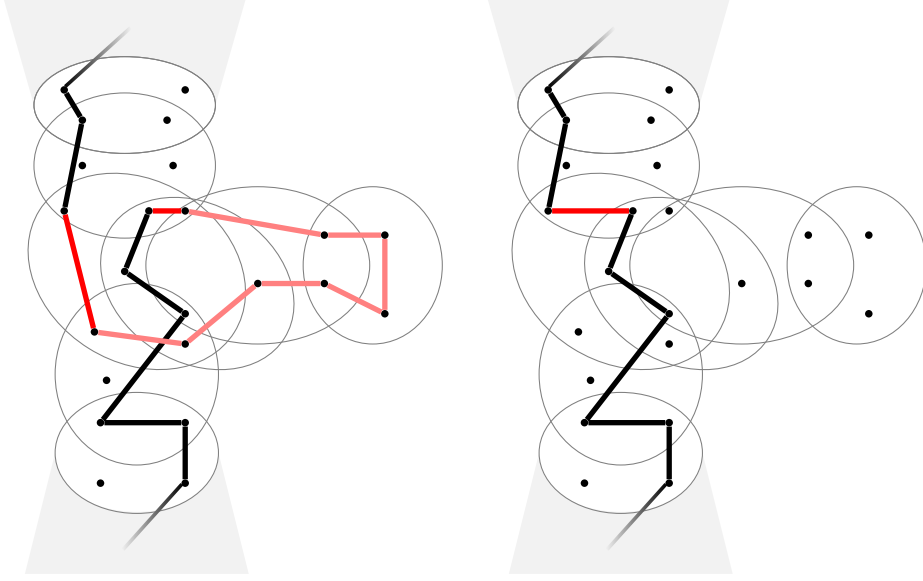
Given a C2RPQ  $\psi$  and a tree decomposition  $\delta$  of  $\psi$ , we say that a path  $\alpha$  in  $\psi$  is *unmeandering* (wrt.  $\delta$ ) if each bag of  $\delta$  covers at most one atom of  $\alpha$ , i.e. it contains at most one such atom with all arguments belonging to the bag.

Consider a C2RPQ  $\psi \in \mathbb{Q}_k(\mathbb{SD}_r(\Phi))$  obtained by quotienting an  $r$ -subdivision of a C2RPQ  $\phi$  from  $\Phi$ . Identifying variables does not break paths, in the sense that  $L$ -paths are still present in  $\psi$ , except that they need not be simple any more. We call a tree decomposition  $\delta$  of  $\psi$  *unmeandering* if each  $L$ -path in  $\psi$  is unmeandering wrt.  $\delta$ .

Let  $\Phi'_{k, \infty}$  be the UC2RPQ obtained from  $\Phi_{k, \infty}$  by keeping only those C2RPQs that admit an unmeandering tree decomposition of width  $k$ , and dropping the remaining ones. By shortcutting meanders as shown in Figure 1, we can refine Lemma 6 as follows.

► **Lemma 7.** *For each UC2RPQ  $\Phi$ ,  $\Phi'_{k, \infty}$  is a TW- $k$  approximation of  $\Phi$ .*

**Proof.** By construction,  $\Phi'_{k, \infty} \subseteq \Phi_{k, \infty} \subseteq \Phi$ . We show that  $\Phi_{k, \infty} \subseteq \Phi'_{k, \infty}$  and thus,  $\Phi \subseteq_k \Phi'_{k, \infty}$ . Let  $\phi'$  be a C2RPQ from  $\Phi_{k, \infty}$ . That is,  $\phi'$  is obtained from a C2RPQ  $\phi$  of  $\Phi$  by an  $r$ -subdivision followed by variable identification, and  $\phi'$  has TW  $k$ . Let  $\delta = (T, \chi)$  be a TW- $k$  decomposition of  $\phi'$  and let  $\mathcal{G}$  be a database such that  $\mathcal{G} \models \phi'$ . We construct a query  $\phi''$  from  $\Phi'_{k, \infty}$  such that  $\mathcal{G} \models \phi''$ . For every atom  $L(u, v)$  in  $\phi$ , query  $\phi'$  has atoms  $L[s_i, s_{i+1}](v_i, v_{i+1})$  for  $0 \leq i < l$ . We obtain  $\phi''$  from  $\phi'$  by merging these atoms exhaustively, as follows. As long



■ **Figure 1** Shortcutting a meander. Whenever two atoms (dark red edges on the left) of some  $L$ -path are covered by the same bag, we replace the whole segment between them (dark and light red edges on the left) with a single summarizing atom (dark red on the right).

as some bag of  $\delta$  covers two atoms of the forms  $L[s_i, s_{i'}](v_i, v_{i'})$  and  $L[s_{j'}, s_j](v_{j'}, v_j)$  with  $i' \leq j'$ , we replace them by  $L[s_i, s_j](v_i, v_j)$ . After each merging step, the current version of  $\phi'$  satisfies the following invariant:

- it has TW  $k$  (a witnessing decomposition is obtained from  $\delta$  by dropping unused variables);
- it is satisfied in  $\mathcal{G}$  (via the same homomorphism restricted to currently used variables);
- it can be obtained from  $\phi$  via an  $r$ -subdivision followed by variable identification; and
- it contains  $L$ -atoms  $L[s_{i_0}, s_{i_1}](v_{i_0}, v_{i_1}), \dots, L[s_{i_m}, s_{i_{m+1}}](v_{i_m}, v_{i_{m+1}})$  for some  $m \leq l$  and  $0 = i_0 < i_1 < \dots < i_{m+1} = l + 1$ .

When no more merges of  $L$  atoms are possible, the resulting query still satisfies the invariant and additionally no bag of  $\delta$  covers more than one of its  $L$ -atoms. We perform this exhaustive procedure for each atom  $L(u, v)$  in  $\phi$ . The resulting query  $\phi''$  belongs to  $\Phi'_{k, \infty}$  and holds in  $\mathcal{G}$ , as required. ◀

We now prove a strong structural property for unmeandering paths, showing that they manifest in tree decompositions in one of three simple ways, illustrated in Figure 2.

► **Lemma 8.** *Consider a C2RPQ  $\psi$ , a tree decomposition  $\delta = (T, \chi)$  of  $\psi$ , and an unmeandering path  $A_0(x_0, x_1), A_1(x_1, x_2), \dots, A_n(x_n, x_{n+1})$  in  $\psi$ .*

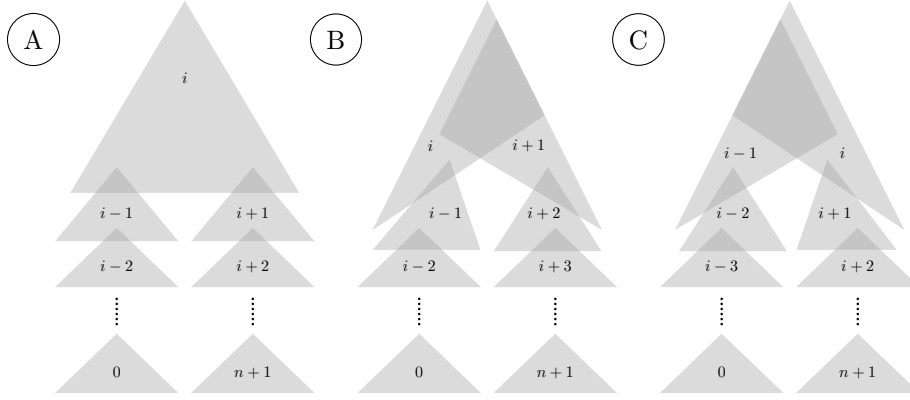
*Then, there exists some  $0 \leq i \leq n + 1$  such that:*

$$\delta(x_0) >_T \delta(x_1) >_T \dots >_T \delta(x_i) \leq_T \delta(x_{i+1}) <_T \delta(x_{i+2}) <_T \dots <_T \delta(x_{n+1})$$

*Moreover, if  $1 \leq i \leq n$ , then either:*

1.  $\delta(x_{i-1}) \not\leq_T \delta(x_{i+1})$  (Figure 2A), or
  2.  $\delta(x_{i-1}) >_T \delta(x_{i+1})$  and  $\delta(x_{i-1}) \not\leq_T \delta(x_j)$  for all  $j > i + 1$  (Figure 2B), or
  3.  $\delta(x_{i-1}) <_T \delta(x_{i+1})$  and  $\delta(x_j) \not\leq_T \delta(x_{i+1})$  for all  $j < i - 1$  (Figure 2C).
- In particular,  $\delta(x_0), \delta(x_1), \dots, \delta(x_{n+1})$  are all different, except potentially  $\delta(x_i)$  and  $\delta(x_{i+1})$ . Also, if  $x_j$  and  $x_{j'}$  belong to the same bag, then  $|j - j'| \leq 1$ .*





■ **Figure 2** An unmeandering path in a tree decomposition. A triangle with label  $j$  represents the subtree of the decomposition consisting of bags containing the  $j$ -th variable of the path.

Note that Lemma 8 implies that each unmeandering path is simple, unless it consists of a single atom. Indeed, while  $\delta(x_i)$  and  $\delta(x_{i+1})$  might be equal,  $x_i = x_{i+1}$  would mean that each bag covering  $A_{i-1}(x_{i-1}, x_i)$  or  $A_{i+1}(x_{i+1}, x_{i+2})$  also covers  $A_i(x_i, x_{i+1})$ .

As the last step before establishing the size bound, we use Lemma 8 to simplify tree decompositions of queries in  $\Phi'_{k,\infty}$ . Given a tree decomposition  $\delta = (T, \chi)$  of a C2RPQ  $\psi$ , and an unmeandering path  $\alpha = A_0(x_0, x_1), \dots, A_n(x_n, x_{n+1})$ , we define the *chevron* of  $\alpha$ , written  $\hat{\alpha}$ , as the union of two shortest paths in  $T$ : from  $\delta(x_0)$  to  $\delta(x_i)$  and from  $\delta(x_i)$  to  $\delta(x_{n+1})$ . By the *main path* of the chevron we mean the shortest path from  $\delta(x_0)$  to  $\delta(x_{n+1})$ . Note that the main path is contained in the chevron and the highest node on the main path is the lowest common ancestor of  $\delta(x_0)$  to  $\delta(x_{n+1})$ . We call it the *key* of the chevron. We refer to  $\delta(x_0)$  and  $\delta(x_{n+1})$  as the *endpoints* of the chevron, and to  $\delta(x_i)$  as the *tip*. We divide the main path into two parts: the *up-path* from  $\delta(x_0)$  to the key, and the *down-path* from the key to  $\delta(x_{n+1})$ . Note that  $\hat{\alpha}$  contains all  $\delta(x_0), \delta(x_1), \dots, \delta(x_{n+1})$ , and each atom  $A_j(x_j, x_{j+1})$  of  $\alpha$  is covered by a bag from the main path of  $\hat{\alpha}$ . Given a C2RPQ  $\psi \in \mathbb{Q}_k(\mathbb{SD}_r(\Phi))$ , a *chevron tree decomposition* of  $\psi$  is any unmeandering tree decomposition in which each bag belongs to the chevron of some  $L$ -path in  $\psi$ .

► **Lemma 9.** *If a C2RPQ  $\psi \in \mathbb{Q}_k(\mathbb{SD}_r(\Phi))$  has an unmeandering tree decomposition of width  $k$ , then it has a chevron tree decomposition of width  $k$ .*

Consider a chevron tree decomposition  $\delta = (T, \chi)$  of a C2RPQ  $\psi \in \mathbb{Q}_k(\mathbb{SD}_r(\Phi))$ . By a *critical* node in  $T$  we mean a leaf, a root, a node that has at least two children, or an endpoint of the chevron of an  $L$ -path. A *segment* in  $\delta$  is a path of the form  $\text{path}_T(u, v)$  for some critical nodes  $u$  and  $v$  such that  $u <_T v$ , and all remaining nodes in  $\text{path}_T(u, v)$  are non-critical. The number of leaves in  $\delta$  is bounded by the number of variables in the original C2RPQ  $\phi \in \Phi$  from which  $\psi$  was obtained, and there are at most twice as many critical nodes. There is no bound, however, on the length of the segments. Our last step in the proof of Theorem 2 essentially consists in providing such a bound. We begin with two preparatory observations, which will be also useful in Section 4.

▷ **Claim 10.** *If a segment  $p$  in  $\delta$  shares a non-critical node with the main path of a chevron  $\hat{\alpha}$  of an  $L$ -path  $\alpha$  in  $\psi$ , then  $p$  is contained in the up-path or the down-path of  $\hat{\alpha}$  (but not both).*

We write  $\mathcal{L}(p)$  for the set of  $L$ -atoms such that segment  $p$  is contained in the main path of the chevron of the  $L$ -path in  $\psi$ . We also define  $\text{dir}_p: \mathcal{L}(p) \rightarrow \{\uparrow, \downarrow\}$  as  $\text{dir}_p(L) = \uparrow$  if  $p$  is contained in the up-path of the chevron of the  $L$ -path, and  $\text{dir}_p(L) = \downarrow$  if  $p$  is contained in the down-path of the chevron of the  $L$ -path.

## 22:10 Evaluating Graph Queries Using Semantic Treewidth

▷ **Claim 11.** Let  $p = \text{path}_T(u, v)$  be a segment in  $\delta$  and  $\alpha = A_0(x_0, x_1), \dots, A_n(x_n, x_{n+1})$  an  $L$ -path in  $\psi$  for some  $L \in \mathcal{L}(p)$ . Then, the atoms of  $\alpha$  covered by bags from  $p$  are

$$A_\ell(x_\ell, x_{\ell+1}), A_{\ell+1}(x_{\ell+1}, x_{\ell+2}), \dots, A_{m-1}(x_{m-1}, x_m)$$

for some  $0 \leq \ell \leq m \leq n + 1$  with  $x_\ell \in \chi(u)$  and  $x_m \in \chi(v)$ , and there are nodes  $u_\ell, u_{\ell+1}, \dots, u_{m-1} \in p$  such that  $u_j$  covers  $A_j(x_j, x_{j+1})$  for all  $\ell \leq j < m$ , and either  $u_\ell <_T u_{\ell+1} <_T \dots <_T u_{m-1}$  or  $u_\ell >_T u_{\ell+1} >_T \dots >_T u_{m-1}$ .

We are now ready for the final step in the proof of Theorem 2. So far we have constructed an infinitary TW- $k$  approximation based on arbitrary subdivisions of atoms in the original UC2RPQ. We next show that it suffices to consider bounded subdivisions. For a given  $r$ , let  $\Phi'_{k,r}$  be the UC2RPQ obtained from  $\Phi'_{k,\infty}$  by keeping only those C2RPQs that belong to  $\mathbb{Q}_k(\mathbb{SD}_r(\Phi))$ , and dropping the remaining ones.

► **Lemma 12.** *For each UC2RPQ  $\Phi$ , there is a positive integer  $r$ , singly exponential in the size of  $\Phi$ , such that  $\Phi'_{k,r}$  is a TW- $k$  approximation of  $\Phi$ .*

**Proof.** We show that  $\Phi'_{k,\infty} \subseteq \Phi'_{k,r}$  for some  $r < \infty$  whose value will follow from the construction. Take  $\psi$  from  $\Phi'_{k,\infty}$  and let  $\delta = (T, \chi)$  be a chevron tree decomposition of  $\psi$ . We transform  $\psi$  into  $\psi'$  from  $\Phi'_{k,r}$  for such that  $\psi \subseteq \psi'$ .

We have already seen that the number of critical nodes in  $\delta$  is bounded linearly in the size of  $\Phi$ , but the segments in  $\delta$  can be arbitrarily long. We obtain  $\psi'$  by shrinking the segments.

Consider a segment  $p$  in  $\delta$  and an internal node  $v$  in  $p$ . We annotate variables in  $v$  with multiple pairs of the form  $(L, s)$ , where  $s$  is a state of the automaton for  $L$ . For each  $L \in \mathcal{L}(p)$  with  $\text{dir}_p(L) = \downarrow$ , we annotate  $x$  with  $(L, s)$  if there is an atom  $L[s, s'](x, x')$  in  $\psi$ . For each  $L \in \mathcal{L}(p)$  with  $\text{dir}_p(L) = \uparrow$ , we annotate  $x$  with  $(L, s)$  if there is an atom  $L[s', s](x', x)$  in  $\psi$ . Let  $\zeta(v)$  be the set of variables in  $\chi(v)$  that have non-empty annotation. We say that internal nodes  $v_1$  and  $v_2$  are *alike* if there is an isomorphism between  $\psi$  restricted to  $\chi(v_1)$  and  $\psi$  restricted to  $\chi(v_2)$  that preserves annotations and is identity over  $\chi(v_1) \cap \chi(v_2)$ . (Note that annotations of the same variable in different bags in  $p$  are the same.)

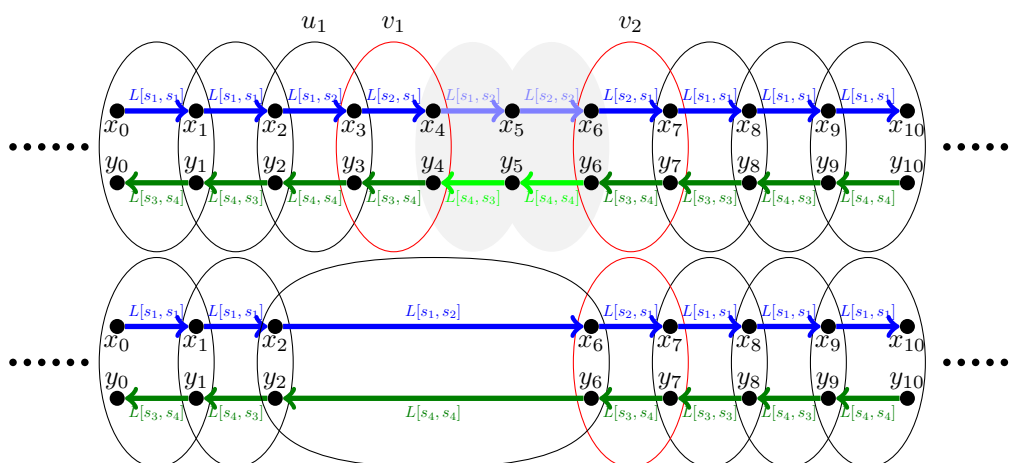
Suppose that segment  $p$  contains two internal nodes  $v_1$  and  $v_2$  such that  $v_1 <_T v_2$  and  $v_1$  and  $v_2$  are alike via an isomorphism  $\iota$ . Let  $u_1$  be the parent of  $v_1$ , and  $u_2$  the parent of  $v_2$ . Note that  $u_1$  and  $u_2$  belong to  $p$ , and  $v_1 \leq_T u_2$ . We can then shrink  $p$  as follows. We replace the subtree of  $T$  rooted at  $v_1$  with the subtree of  $T$  rooted at  $v_2$ , and replace each variable  $x \in \chi(u_1) \cap \chi(v_1)$  with  $\iota(x)$ , both in the tree decomposition and in the query. Note that this removes all bags from  $\text{path}_T(v_1, u_2)$ . We modify the query accordingly: we remove all atoms that use a variable that does not occur in the modified tree decomposition.

Recalling the characterization of Claim 11, we can see that the effect this has on the query is that for each  $L \in \mathcal{L}(p)$ , a subpath

$$L_j[s_j, s_{j+1}](x_j, x_{j+1}), L_{j+1}[s_{j+1}, s_{j+2}](x_{j+1}, x_{j+2}), \dots, L_{k-1}[s_{k-1}, s_k](x_{k-1}, x_k)$$

with  $s_{j+1} = s_k$  gets replaced with  $L_j[s_j, s_k](x_j, x_k)$ , as shown in Figure 3. Hence, after a single shrinking, the query still belongs to  $\mathbb{Q}_k(\mathbb{SD}_r(\Phi))$ . Also, the modified tree decomposition has still width at most  $k$  and is unmeandering. Hence, the modified query belongs to  $\Phi'_{k,\infty}$ . Finally, if the original query holds in some database, so does the modified one, with the same witnessing mapping of variables to nodes.

We obtain  $\psi'$  by repeatedly shrinking segments until it is no longer possible. By the above invariants,  $\psi' \in \Phi'_{k,\infty}$  and  $\psi \subseteq \psi'$ . It remains to see that  $\psi' \in \mathbb{Q}_k(\mathbb{SD}_r(\Phi))$  for some  $r$  exponential in the size of  $\Phi$ . Because we the tree decomposition corresponding to  $\psi'$



■ **Figure 3** Shrinking a segment.

is unmeandering, the number of atoms in each  $L$ -path is bounded by the size of the tree decomposition. We already know that the number of critical nodes is linear in the size of  $\Phi$ . Hence, so is the number of segments. After shrinking, no two internal bags in a segment are alike, so the length of each segment is bounded by a function single exponential in the size of  $\Phi$ . Hence, the whole tree decomposition is of size single exponential in the size of  $\Phi$ , and we are done. ◀

Lemma 12 yields a  $TW$ - $k$  approximation of a UC2RPQ  $\Phi$  in which each C2RPQ has size at most singly exponential in the size of  $\Phi$ . As there are at most doubly exponentially many such C2RPQs, Theorem 2 follows directly from the lemma.

## Related Work

As explained in the beginning of this section,  $TW$ - $k$  approximations for UC2RPQs have already been constructed in [12]. Our construction has some common points, but also differences, to the construction from [12]. In both approaches, an infinitary approximation is constructed, which is first normalized, and then is shrunk by bounding the height of tree decompositions. We will address these below.

**Normalization of the approximation.** In our case the normalization step consists in constructing approximations which admit unmeandering tree decompositions, while in [12] original paths are contracted such that a bag of the tree decomposition is not revisited once it is left. However, inside bags, atoms belonging to original paths might form cyclic paths and they are not contracted. This is achieved by so-called *tagged tree decompositions*. As we will see in the next section, the notion of unmeandering tree decomposition will be crucial for our algorithm to efficiently evaluate UC2RPQs. As part of the algorithm, we will define a notion of types which capture the content of bags that occur in such decompositions. While we do not preclude the possibility to define a similar notion based on decompositions with arbitrary bags, it would be much more cumbersome.

**Shrinking of the approximation.** Both approaches shrink paths of unbounded length in tree decompositions of approximations. In our approach we annotate bags of tree decompositions and completely remove paths between nodes which are alike w.r.t. annotations. Ferreira

and Morvan [12] annotate bags as well, using a less-constrained notion for identifying similar bags. However, due to this less-constrained notion, it is not possible for them to fully remove the path between such nodes: instead, by using *nice tree decompositions*, they shrink the path between two such similar nodes to a path of constant length.

#### 4 Exploiting Semantic Treewidth

As discussed in the introduction, it is already known from [6] that UC2RPQs of bounded TW are fpt. There, an algorithm based on computing and evaluating a witness of bounded TW is presented. However, the treewidth of the witness might not be optimal: for a UC2RPQ  $\Phi$  of semantic TW- $k$ , the witness query  $\Phi_w$  from [6] might have treewidth  $2k + 1$ . Using standard results concerning CQ evaluation [15] and RPQ evaluation [7], it is possible to evaluate  $\Phi_w$  in time  $O(p(|\Phi_w|)|\mathcal{G}|^{2k+2})$ , for some polynomial  $p$ : one can first “materialize” all the RPQs which occur in C2RPQs in  $\Phi_w$  in the database, i.e. create a new database  $\mathcal{G}'$  which contains all facts  $L(a_1, a_2)$  such that  $\mathcal{G} \models L(a_1, a_2)$ , for every atom of the form  $L(u, v)$  in  $\Phi_w$ . This can be done in polynomial time in the size of  $\Phi_w$  and  $\text{dom}(\mathcal{G}') = \text{dom}(\mathcal{G})$ . In a second step,  $\Phi_w$  can be seen as a CQ of TW  $2k + 1$  over  $\mathcal{G}'$ , which can then be evaluated in time  $O(p(|\Phi_w|)|\mathcal{G}'|^{2k+2})$ , where  $p$  is some polynomial. As  $|\Phi_w|$  is bounded by a singly exponential function in  $|\Phi|$ , one obtains an algorithm for evaluating  $\Phi$  which runs in time  $O(g(|\Phi|)|\mathcal{G}|^{2k+2})$ , for  $g$  some singly exponential function.

On the other hand, similarly to [12], in Section 3, for every  $k > 1$ , we constructed semantic treewidth witnesses with optimal TW in the form of TW  $k$ -approximations of size doubly exponential in the size of  $\Phi$ . Let  $\Phi_k$  be such an approximation of a UC2RPQ  $\Phi$ . Using similar arguments as for  $\Phi_w$ , we obtain that there must be some function  $g$  such that  $\Phi_k$  can be evaluated in time  $O(g(|\Phi|)|\mathcal{G}|^{k+1})$ , with  $g$  a doubly exponential function. While in our parameterized setting, where we expect the size of the data to be much larger than that of the query, this is an important improvement compared to the approach based on the witness of non-optimal TW  $2k + 1$ , the doubly exponential combined complexity makes such an approach prohibitive.

However, as we will show in the following, there is an algorithm for evaluating  $\Phi$  in time  $O(g(|\Phi|)|\mathcal{G}|^{k+1})$ , with  $g$  a singly exponential function. The evaluation procedure uses  $\Phi'_{k,\infty}$ , the infinitary UC2RPQ constructed in Section 3, Lemma 7, which has the nice structural property that all its C2RPQs admit unmeandering tree decompositions. We first observe that  $\Phi'_{k,\infty}$  can be regarded as a union of singly exponentially many queries, where each of these queries is an infinitary UC2RPQ which contains all C2RPQs from  $\Phi'_{k,\infty}$  which share some common structure, called *skeleton*. We achieve a compact representation of each such UC2RPQ by means of so-called *reachability queries* which are queries which impose reachability conditions from one  $k + 1$ -tuple of variables to another  $k + 1$ -tuple. In this view a *skeleton query* is a tree of reachability queries, one query associated to each edge of the tree. This makes it possible to encode the evaluation of such a query in a Datalog program which simulates a bottom-up evaluation of the skeleton tree. As the program has width  $k + 1$  and singly exponential size, we obtain the above-mentioned complexity bounds.

#### Abstracting C2RPQs via Skeletons

Recall that each C2RPQ  $\phi'$  from  $\Phi'_{k,\infty}$  is from  $\mathbb{Q}_k(\mathbb{SD}_l(\phi))$ , for some  $\phi \in \Phi$ , and  $l > 0$  and, thus there exists a natural mapping  $h$  from  $\text{var}(\phi)$  to  $\text{var}(\phi')$ . Given a tree decomposition  $\delta'$  for  $\phi'$  one can define the set of critical nodes and segments of  $\delta'$ , denoted here as  $\text{crit}(\delta')$ , and  $\text{seg}(\delta')$ , as in Section 3. A *segment signature*  $\sigma$  is a tuple of the form  $(\mathcal{L}_\sigma, \text{dir}_\sigma)$ , where  $\mathcal{L}_\sigma$  is

a set of regular expressions and  $\text{dir}_\sigma: \mathcal{L}_\sigma \rightarrow \{\uparrow, \downarrow\}$ . For every segment  $p \in \text{seg}(\delta')$ , we denote with  $\text{sig}(p)$  its segment signature defined as follows:  $\mathcal{L}_{\text{sig}(p)} = \mathcal{L}(p)$  and  $\text{dir}_{\text{sig}(p)}(L) = \text{dir}_p(L)$ , for every  $L \in \mathcal{L}_{\text{sig}(p)}$ . We denote with  $\text{sig}(\delta')$  the set of all segment signatures  $\text{sig}(p)$ , where  $p \in \text{seg}(\delta')$  and with  $\text{sig}(\Phi'_{k,\infty})$  the union of all sets  $\text{sig}(\delta')$ , where  $\delta'$  is an unmeandering tree decomposition of some C2RPQ  $\phi'$  from  $\Phi'_{k,\infty}$ .

We introduce some further notations: an *extended*<sup>1</sup> C2RPQ is a C2RPQ in which we allow also atoms of the form  $L[s](v)$ , where  $L$  is a regular expression and  $s \in \text{states}(A(L))$ . For a C2RPQ  $\theta$ , we denote with  $\text{ext}(\theta)$  the extended C2RPQ obtained from  $\theta$  by adding atoms  $L[s](v)$  and  $L[s'](v')$ , for every atom of the form  $L[s, s'](v, v')$  in  $\theta$ . Conversely, for an extended C2RPQ  $\gamma$ , we denote with  $\text{bin}(\gamma)$ , the C2RPQ obtained from  $\gamma$  by maintaining only binary atoms.

The notation  $\text{var}$  is lifted to extended C2RPQ as expected. Also, given an extended C2RPQ  $\theta$  and a set of variables  $\mathcal{V} \subseteq \text{var}(\theta)$ , we denote with  $\theta_{\mathcal{V}}$  the extended C2RPQ obtained from  $\theta$  by retaining every atom from  $\theta$  with all arguments from  $\mathcal{V}$  (be it binary or unary). Given a set of regular expressions  $\mathcal{L}$ , and a set of variables  $\mathcal{V}$ ,  $\mathbb{C}_{k+1}^{\mathcal{L}}(\mathcal{V})$  is the set of all extended C2RPQs using regular expressions from  $\mathcal{L}$  with at most  $k+1$  variables from  $\mathcal{V}$ .

We are now ready to define the data structure which will serve as a common abstraction for different C2RPQs from  $\Phi_{k,\infty}$ :

► **Definition 13.** *Given  $\phi' \in \Phi_{k,\infty}$  and  $\delta' = (T', \chi')$  a tree decomposition of  $\phi'$ , the skeleton of  $\phi'$  w.r.t.  $\delta'$ ,  $\text{sk}(\phi', \delta')$ , is a triple  $(T, q, \mu)$ , where:*

1.  $T = (\text{crit}(\delta'), E)$  is the tree induced by  $<^T_T$  on  $\text{crit}(\delta')$ ;
2.  $q: \text{crit}(\delta') \rightarrow \mathbb{C}_{k+1}^{\mathcal{L}_{\phi'}}(\text{var}(\phi'))$  is the function:  $q(v) = \text{ext}(\phi')_{\chi'(v)}$ , for  $v \in \text{crit}(\delta')$ ; and
3.  $\mu: E \rightarrow \text{sig}(\delta')$  is the function:  $\mu(v_1, v_2) = \text{sig}(v_2, v_1)$ , for  $(v_1, v_2) \in E$ .

Intuitively,  $\text{sk}(\phi', \delta')$  abstracts  $\phi'$  w.r.t.  $\delta'$  by keeping only atoms covered by critical bags (endpoints of segments) together with new unary atoms which are pointers to the intersection(s) of  $L$ -paths with such a bag; the inner parts of segments are replaced with their signature (captured by edges from  $E$  labelled by  $\mu$ ).

Let  $\text{sk}(\Phi'_{k,\infty})$  be the set of all  $\text{sk}(\phi', \delta')$ , where  $\phi'$  is a C2RPQ from  $\Phi'_{k,\infty}$  and  $\delta'$  is an unmeandering tree decomposition of  $\phi'$ . For  $sk \in \text{sk}(\Phi'_{k,\infty})$ , and  $\phi'$  from  $\Phi'_{k,\infty}$ , we say that  $\phi'$  *abstracts to*  $sk$  if there exists an unmeandering tree decomposition  $\delta'$  such that  $\text{sk}(\phi', \delta') = sk$ . In the following, for every  $sk \in \text{sk}(\Phi'_{k,\infty})$ , we let  $\Phi_{sk}$  be the UC2RPQ which is the union of all C2RPQs from  $\Phi'_{k,\infty}$  which abstract to  $sk$ . Thus,  $\Phi_{sk}$  contains all C2RPQs which share common structure in the form of  $sk$ .

As all disjuncts of  $\Phi'_{k,\infty}$  are covered by some UC2RPQ of the form  $\Phi_{sk}$ , with  $sk \in \text{sk}(\Phi'_{k,\infty})$ , we have that:

$$\Phi'_{k,\infty} = \bigvee_{sk \in \text{sk}(\Phi'_{k,\infty})} \Phi_{sk}.$$

► **Proposition 14.** *The set  $\text{sk}(\Phi'_{k,\infty})$  is of size singly exponential in  $|\Phi|$  and can be computed in singly exponential time in  $|\Phi|$ .*

In the following, we show how each query  $\Phi_{sk}$  can be reformulated into a query which mirrors the structure of a skeleton, in the sense that it can be seen as a conjunction of queries, one corresponding to each edge in the tree underlying the skeleton. Each such query can be seen as a type reachability query, i.e. a query which specifies how to reach the top type induced by the segment signature attached to an edge from the corresponding bottom type.

<sup>1</sup> We will not define the semantics of such a C2RPQ as its purpose is to serve mainly as a syntactical object for later constructions.

## Types and Reachability Queries

We start by defining  $\sigma$ -types over at most  $k + 1$  variables, where  $\sigma$  is a segment signature. Intuitively, such a type captures for every  $L \in \mathcal{L}_\sigma$  the intersection of the  $L$ -path with a bag in some unmeandering tree decomposition belonging to a  $\sigma$ -segment, be it either in the form of an  $L$ -atom or a single variable which is visited by the path in a certain state. We use extended C2RPQs to capture such information. In the following, we reserve a set of fresh  $2k + 1$  variables  $\mathbb{V} = \{v_1, \dots, v_{2k+1}\}$ .

► **Definition 15.** For a segment signature  $\sigma$ , a  $\sigma$ -type  $\tau$  is an extended full C2RPQ from  $\mathbf{C}_{k+1}^{\mathcal{L}_\sigma}(\mathbb{V})$  such that for every  $L \in \mathcal{L}_\sigma$ , one of the following holds:

1.  $\tau$  contains exactly one atom of the form  $L[s, s'](v, v')$ ;
2.  $\tau$  contains exactly one atom of the form  $L[s](v)$ .

Given such a  $\sigma$ -type  $\tau$ , based on the direction of paths in  $\sigma$ , we can compute enter and exit points in the form of pairs (variable, state) for each path in  $\sigma$  w.r.t.  $\tau$ . To this purpose, we define functions  $\text{in}_\tau: \mathcal{L}_\sigma \rightarrow \text{var}(\tau) \times \bigcup_{L \in \mathcal{L}_\sigma} \text{states}(A(L))$  and  $\text{out}_\tau: \mathcal{L}_\sigma \rightarrow \text{var}(\tau) \times \bigcup_{L \in \mathcal{L}_\sigma} \text{states}(A(L))$  as follows:

1. if  $\tau$  contains an  $L$ -atom of the form  $L[s, s'](v, v')$  and  $\text{dir}_s(L) = \uparrow$ , then  $\text{in}_\tau(L) = (v, s)$  and  $\text{out}_\tau(L) = (v', s')$ ;
2. if  $\tau$  contains an  $L$ -atom of the form  $L[s, s'](v, v')$  and  $\text{dir}_s(L) = \downarrow$ , then  $\text{in}_\tau(L) = (v', s')$  and  $\text{out}_\tau(L) = (v, s)$ ;
3. if  $\tau$  contains no  $L$ -atom of the form  $L[s, s'](v, v')$ , then  $\text{in}_\tau(L) = \text{out}_\tau(L) = (v, s)$ , where  $L[s](v)$  is the unique  $L$ -atom from  $\tau$ .

Conditions (1) and (2) above can be explained as follows. To build segments, types have to be matched. With “out” we single out the node and state from which the path should be continued in the next type and with “in” the node and state which continues the path from a previous type; this is dependent on the direction of the path relative to the segment it traverses. Because segments are built bottom-up, upward paths are built forward and downward paths are built backwards. That is, for downward paths, the first argument of an atom will have to be matched when we stack up another type; for upward paths, the second. Constraint (3) simply ensures that paths are carried over in  $\sigma$ -types; it might be the case that nothing new is added by a type to a certain path, in which case both “in” and “out” refer to the same variable and the same state.

Building on these functions, we introduce a notion of type compatibility which captures the idea that a  $\sigma$ -type  $\tau_1$  is compatible with a  $\sigma$ -type  $\tau_2$  if  $\tau_1$  can be continued by  $\tau_2$ .

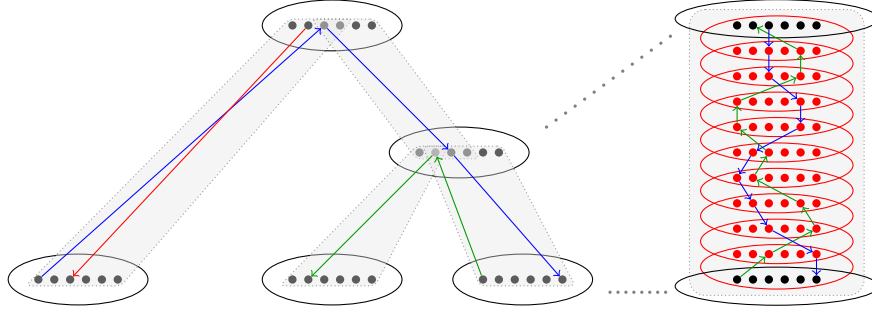
► **Definition 16.** Given a segment signature  $\sigma$ , and two  $\sigma$ -types  $\tau_1$  and  $\tau_2$ ,  $\tau_1$  is compatible with  $\tau_2$  if for every  $L \in \mathcal{L}_\sigma$  it is the case that either:

1. there is some atom of the form  $L[s, s'](v_1, v_2) \in \tau_2 \cap \tau_1$ , or
2.  $\text{in}_{\tau_2}(L) = \text{out}_{\tau_1}(L)$ ;

Furthermore, for every  $v \in \text{var}(\tau_1) \cap \text{var}(\tau_2)$ , it must be the case that there exists some  $L \in \mathcal{L}_\sigma$  such that either  $v = v_1$  or  $v = v_2$  as in condition (1) above, or  $\text{in}_{\tau_2}(L) = \text{out}_{\tau_1}(L) = v$ .

Thus, for a type  $\tau_1$  to be continuable by a type  $\tau_2$  it must be the case that they either share a common part of a path in the form of an  $L$ -atom or the ingoing  $L$ -variable of the second type  $\tau_2$  coincides with the outgoing  $L$ -variable of the first type  $\tau_1$ . Also, only variables which occur in such a shared atom or are used to continue  $L$ -paths from one type to the next, occur in both types.





■ **Figure 4** Constructing a skeleton query.

We will next introduce the notion of reachability query between so-called  $\sigma$ -types instantiations. Given a type  $\tau$  and a tuple of variables  $\vec{u}$  such that  $|\vec{u}| = |\text{var}(\tau)|$  (we assume that all variables in  $\vec{u}$  are distinct), the *instantiation of  $\tau$  with  $\vec{u}$* ,  $\tau(\vec{u})$ , is the extended C2RPQ obtained from  $\tau$  by replacing  $\text{var}(\tau)$  with  $\vec{u}$ .

► **Definition 17.** Given two  $\sigma$ -types instantiations  $\tau'_1(\mathbf{x}_1)$  and  $\tau'_2(\mathbf{x}_2)$ , a reachability query from  $\tau'_1(\mathbf{x}_1)$  to  $\tau'_2(\mathbf{x}_2)$ , denoted  $\text{reach}(\tau'_1(\mathbf{x}_1), \tau'_2(\mathbf{x}_2))$  is the union of all C2RPQs of the form

$$\exists \mathbf{y} \bigwedge_{0 \leq i \leq m} \text{bin}(\tau_i(\mathbf{y}_i)),$$

where  $m \in \mathbb{N}$ ,  $m \geq 2$ , and  $(\tau_i(\mathbf{y}_i))_{1 \leq i \leq m}$  is a sequence of type instantiations such that:

1.  $\tau_0(\mathbf{y}_0)$  is  $\tau'_1(\mathbf{x}_1)$  and  $\tau_m(\mathbf{y}_m)$  is  $\tau'_2(\mathbf{x}_2)$ ;
2. for every  $0 \leq i < m$ :  $\tau_i$  is compatible with  $\tau_{i+1}$ ;
3. for every  $i \neq j$ ,  $1 \leq l \leq |\mathbf{y}_i|$ , and  $1 \leq n \leq |\mathbf{y}_j|$ :  $\mathbf{y}_{i,l} = \mathbf{y}_{j,n}$  iff  $\mathbf{z}_{i,l} = \mathbf{z}_{j,n}$  where  $\mathbf{z}_i = \text{var}(\tau_i)$  and  $\mathbf{z}_j = \text{var}(\tau_j)$ , and
4.  $\mathbf{y} = \bigcup_{0 < i < m} \mathbf{y}_i \setminus (\mathbf{x}_1 \cup \mathbf{x}_2)$ .

Thus, the reachability query is a UC2RPQ with free variables  $\mathbf{x}_1 \cup \mathbf{x}_2$ , in which each C2RPQ can be seen as the conjunction of C2RPQs occurring in type instantiations from a sequence of type instantiations. Condition (3) in Definition 17 asks for tuples  $\mathbf{y}_i$  and  $\mathbf{y}_j$  to have the same intersection profile as the tuples of variables pertaining to types  $\tau_i$  and  $\tau_j$ ; for example, if  $\text{var}(\tau_i) = (v_1, v_2, v_3)$  and  $\text{var}(\tau_j) = (v_2, v_3, v_4)$ , then  $\mathbf{y}_i$  and  $\mathbf{y}_j$  will be of the form  $(\_, y_1, y_2)$  and  $(y_1, y_2, \_)$ , respectively, and  $\mathbf{y}_i \cap \mathbf{y}_j = \{y_1, y_2\}$ .

## Skeleton Queries

We now return to  $sk = (T, q, \mu)$  from  $\text{sk}(\Phi'_{k,\infty})$ . We associate to every edge  $e = (v_1, v_2) \in E$  a reachability query between the two  $\mu(e)$ -type instantiations induced by  $v_1$  and  $v_2$ . Recall that for every  $v \in V$ ,  $q(v)$  is an extended C2RPQ which contains atoms of the form  $L[s, s'](u, u')$  or  $L[s](u)$ . For  $e = (v_1, v_2) \in E$  and  $i \in \{1, 2\}$ , we can read  $\mu(e)$ -type instantiations  $\tau_e^i(\mathbf{x}_e^i)$  from  $q(v_i)$  by simply maintaining all  $L$ -atoms with  $L \in \mathcal{L}_{\mu(e)}$ .

Let  $\text{reach}_e(\mathbf{x}_e^2, \mathbf{x}_e^1)$  be an abbreviation for  $\text{reach}_e(\tau_e^2(\mathbf{x}_e^2), \tau_e^1(\mathbf{x}_e^1))$  and let  $q_{sk}$  be the following Boolean query:

$$\exists_{e \in E, i \in \{1, 2\}} \mathbf{x}_e^i \bigwedge_{e \in E} \text{reach}_e(\mathbf{x}_e^2, \mathbf{x}_e^1).$$



Figure 4 describes the construction of query  $q_{sk}$ . On the left hand side, it depicts the underlying skeleton  $sk$ : circles denote nodes from  $V$  with their corresponding extended C2RPQs, while the grey quadrangles together with the colored directed paths denote segment signatures. On the right hand side, we show how a C2RPQ from a reachability query corresponding to an edge in the skeleton looks like.

An important observation is the following:

► **Observation 18.** *For every  $sk \in \text{sk}(\Phi'_{k,\infty})$ , it holds that:  $q_{sk} \equiv \Phi_{sk}$ .*

**Proof.** The query  $q_{sk}$  is a conjunction of reachability queries, thus a conjunction of disjunctions of C2RPQs. By distributing the disjunction over conjunction one obtains an equivalent UC2RPQ  $\Theta$  in which each C2RPQ is from  $\Phi'_{k,\infty}$ , and, in particular, from  $\Phi_{sk}$  (as it abstracts to  $sk$ ). Conversely, each C2RPQ which abstracts to  $sk$  can be seen as a conjunction of realizations of reachability queries over segments and thus is from  $\Theta$ . ◀

As a corollary, we obtain the following:

► **Corollary 19.** *For every  $k > 1$ , it is the case that  $\Phi'_{k,\infty} \equiv \bigvee_{sk \in \text{sk}(\Phi'_{k,\infty})} q_{sk}$ .*

Using this insight, it is possible to evaluate queries of the form  $\Phi_{sk}$  by means of a dynamic programming approach based on traversing the data structure  $sk$ . The approach can be seen as a Yannakakis-style [24] bottom-up evaluation algorithm for  $q_{sk}$  in which we have oracles to compute the upper end of a segment/reachability query given its lower end. We encode both parts of the procedure, the meta-level, where segments are combined, and the segment-specific type reachability queries, as a joint  $(k+1)$ -Datalog program  $\Pi_{sk}$ .

## Datalog Encoding

We are now ready to define a Datalog rewriting  $\Pi_{\Phi,k}$  for  $\Phi_{\infty,k}$ . The program will have several components. First, we assume that we have a program  $\Pi_{\mathcal{L}}$  which contains binary IDBs  $L$  (we slightly overload the notation, to avoid introducing new names), one for each atom  $L(u,v)$  occurring in  $\Phi_{\infty,k}$ . The purpose of the program is to materialize such atoms w.r.t. the database. The encoding is fairly standard using rules with only two variables, so we do not provide it here [2].

Next we define for every segment signature  $\sigma$ , a program  $\Pi_{\sigma}$  which captures  $\sigma$ -types and the way they can be interlinked to build segments. We start by introducing IDBs corresponding to types. For every  $\sigma$ -type  $\tau$ , we will have an IDB of the form  $I[\tau]$  with arity  $|\text{var}(\tau)|$ . Furthermore, for every such type we denote with  $\text{exit}_{\tau}$  the set of all variables from  $\text{var}(\tau)$  which are exit variable for some path  $L \in \mathcal{L}(p)$ , i.e. there exist some state  $s$  such that  $\text{out}_{\tau}(L) = (v,s)$ . Then, for such a type and every set of variables  $\mathbf{v}'$  such that  $\text{exit}(\tau) \subseteq \mathbf{v}' \subseteq \text{var}(\tau)$  we will have an IDB predicate  $I[\tau, \mathbf{v}']$  of arity  $|\mathbf{v}'|$ . Intuitively, the predicate stands for projections of a type; it collects instances of the (sub-)type in the database. For every set of variables  $\mathbf{v}'$  such that  $\text{exit}(\tau) \subseteq \mathbf{v}' \subseteq \text{var}(\tau)$ , program  $\Pi_{\sigma}$  contains a rule of the form:

$$I[\tau](\text{var}(\tau)) \rightarrow I[\tau, \mathbf{v}'](\mathbf{v}') \quad (1)$$

We next provide a rule for combining  $\sigma$ -types. For every two  $\sigma$ -types  $\tau_1$  and  $\tau_2$  such that  $\tau_1$  is compatible with  $\tau_2$ , let  $\mathbf{v} = \text{var}(\tau_1) \cap \text{var}(\tau_2)$ . Then we add the following rule to  $\Pi_{\sigma}$ :

$$I[\tau_1, \mathbf{v}](\mathbf{v}), \text{bin}(\tau_2) \rightarrow I[\tau_2](\text{var}(\tau_2)) \quad (2)$$

Intuitively, to keep the width of the program bounded by  $k+1$ , when combining compatible types, we first project the first type on the variables relevant for matching, i.e. those variables which occur also in the second type. We remark that although  $\text{var}(\text{bin}(\tau_2))$  might not contain all variables from  $\text{var}(\tau_2)$ , the conditions on type compatibility ensure that  $\mathbf{v} \cup \text{var}(\text{bin}(\tau_2)) = \text{var}(\tau_2)$ . In particular,  $\mathbf{v}$  contains all those variables from  $\text{var}(\tau_2)$  for which there is no  $L$ -atom in  $\text{bin}(\tau_2)$ , but are used to carry over information about  $L$ -paths by means of unary  $L$ -atoms in  $\tau_2$ .

We next provide for every  $sk \in \text{sk}(\Phi_{k,\infty})$ , rules which, building on the axiomatizations of previously introduced IDBs, encode the query  $\Phi_{sk}$ , by simulating a bottom-up evaluation of the equivalent query  $q_{sk}$ . We denote the set of such rules as  $\Pi_{sk}$ .

Let  $sk = (T, q, \mu)$  with  $T = (V, E)$  and  $r$  being the root of  $T$ . Recall that each edge  $e = (v_1, v_2)$  in  $E$  stands for a type reachability query  $\text{reach}_e(\tau_e^2(\mathbf{x}_e^2), \tau_e^1(\mathbf{x}_e^1))$ , where  $\tau_e^i$  is the type instantiation induced by  $\mu(e)$  and  $q(v_i)$ , for  $i \in \{1, 2\}$ . Each node  $v \in V$  is in the scope of several such type reachability queries: some correspond to edges of the form  $(v', v)$ , and at most one such a query corresponds to an edge of the form  $(v, v')$ . For a node  $v$  which is not a leaf, we denote with  $E_v$  the set of edges of the form  $(v', v)$  and with  $\text{low}(v)$  the set of type instantiations corresponding to the second argument of such an edge:  $\{\tau_e^2(\mathbf{x}_e^2) \mid e \in E_v\}$ . Also, for a node  $v$  which is not the root, let  $e_v$  be the unique edge  $(v, v')$  and  $\tau_v(\mathbf{x}_v)$  be the type instantiation corresponding to the first argument of  $e_v$  (i.e. an abbreviation for  $\tau_{e_v}^1(\mathbf{x}_{e_v}^1)$ ).

There are three types of rules in the definition of  $\Pi_{sk}$ : those pertaining to leaves of  $T$ , one rule pertaining to the root, and rules pertaining to the other nodes. We start with the first type. For every node  $v \in V$  which is a leaf of  $T$ , we define a rule which seeds the type instantiation  $\tau_v(\mathbf{x}_v)$ . Note that there might be variables in  $\mathbf{x}_v$  which do not occur in  $\text{bin}(\tau_v(\mathbf{x}_v))$ . To initialize such variables we assume that we have a built-in unary predicate  $\text{dom}()$  which binds a variable to the domain of the given database. Assuming that  $v_1, \dots, v_l$  are those dangling variables, our rule is as follows:

$$\text{bin}(\tau_v(\mathbf{x}_v)), \text{dom}(v_1), \dots, \text{dom}(v_l) \rightarrow I[\tau_v](\mathbf{x}_v) \quad (3)$$

We move on to nodes  $v \in V$  which are neither leaves, nor the root of  $T$ . For such nodes, both  $\tau_v(\mathbf{x}_v)$  and  $\text{low}(v)$  are defined. Assuming  $\text{low}(v) = \{\tau_1(\mathbf{x}_1), \dots, \tau_l(\mathbf{x}_l)\}$  we add the following rule to  $\Pi_{sk}$ :

$$I[\tau_1](\mathbf{x}_1), \dots, I[\tau_l](\mathbf{x}_l), \text{bin}(\tau_v(\mathbf{x}_v)) \rightarrow I[\tau_v](\mathbf{x}_v) \quad (4)$$

Rule 4 intersects the IDBs corresponding to the top ends of segments coming from below to populate the IDB corresponding to the bottom end of the segment going upwards. Finally, we move on to the root node of  $T$ ,  $r$ . Assuming  $\text{low}(r) = \{\tau_1(\mathbf{x}_1), \dots, \tau_l(\mathbf{x}_l)\}$  we add the following rule to  $\Pi_{sk}$ :

$$I[\tau_1](\mathbf{x}_1), \dots, I[\tau_l](\mathbf{x}_l) \rightarrow \text{true} \quad (5)$$

Note that rules of type (4) and (5) as above have width at most  $k+1$  as all  $\mathbf{x}_i$ -s are from some  $\text{var}(q(v))$ , with  $v \in V$ .

Finally, let

$$\Pi_{\Phi,k} = \Pi_{\mathcal{L}} \cup \bigcup_{\sigma \in \text{sig}(\Phi'_{k,\infty})} \Pi_{\sigma} \cup \bigcup_{sk \in \text{sk}(\Phi'_{k,\infty})} \Pi_{sk}$$

The following proposition follows from Proposition 14 and the fact that there are singly exponentially many segment signatures, and for each segment signature  $\sigma$ , singly exponentially many  $\sigma$ -types:

► **Proposition 20.** *For  $\Phi$  a Boolean UC2RPQ, and for every  $k > 1$ , the Datalog program  $\Pi_{\Phi,k}$  is of width  $k + 1$ , has size singly exponential in  $|\Phi|$  and can be constructed in singly exponential time in  $|\Phi|$ .*

Further on, we show that the program  $\Pi_{\Phi,k}$  encodes TW  $k$  approximations:

► **Proposition 21.** *For  $\Phi$  a Boolean UC2RPQ,  $k > 1$ , and  $\mathcal{G}$  be a graph database, it is the case that:  $\mathcal{G} \models \Phi'_{k,\infty}$  iff  $(\mathcal{G}, \Pi_{\Phi,k}) \models \text{true}$ .*

Proposition 20 and Proposition 21 together with complexity results on evaluation of Datalog programs of width  $k + 1$ , yield our main result:

► **Theorem 22.** *Let  $\mathcal{G}$  be a graph database and  $\Phi$  be a Boolean UC2RPQ of semantic TW  $k$  with  $k > 1$ . Then,  $\Phi$  be evaluated in time  $O(f(|\Phi|)|\mathcal{G}|^{k+1})$ , where  $f$  is a singly exponential function.*

## Related Work

The skeleton queries we constructed in this section have connections both to the witness queries  $\Phi_w$  constructed in [20] discussed at the beginning of this section and to the so-called *summary queries* introduced in [12].

Figureira and Morvan [12] introduce so-called *summary queries* which are exponentially more succinct representations of their TW- $k$  approximations. To this purpose, they introduce so-called *path- $l$  approximations* which are queries whose semantics is defined in terms of infinitary C2RPQs which admit a path- $l$  decomposition, where a path decomposition is defined to be any tree decomposition whose underlying tree is a path. Our notions of skeleton queries and type reachability queries are based on similar intuitions, with the difference that our queries are restricted to those which admit unmeandering tree decompositions, and thus we have a clear notion of types and how they can be chained in a type reachability query.

Romero et. al [20] essentially construct witness queries of TW  $2k + 1$  by first constructing infinitary TW- $k$  approximations using sub-divisions and quotients. In a subsequent step, they identify a set of nodes which correspond to our set of critical nodes, and contract paths by considering only their intersection with this set of critical nodes. By virtue of this contraction operation, they obtain tree decompositions of TW  $2k + 1$ : intuitively, each two critical nodes which are top and bottom end of some segment are merged giving rise to a bag in the new tree decomposition. Each of our skeleton queries can be seen as the TW- $k$  approximation of such a C2RPQ of TW  $2k + 1$  belonging to  $\Phi_w$ , as instead of contracting the paths between two critical nodes, we consider all their realizations via paths of width  $k$ .

## 5 Summary and Outlook

In this paper we looked at the problem of efficient evaluation of UC2RPQs of bounded semantic treewidth. Previous approaches based on computing a witness of semantic TW  $k$  of doubly potential size were running in time  $O(f(|\Phi|)|\mathcal{G}|^{k+1})$ , with  $f$  a doubly exponential function, where  $\Phi$  is the size of the UC2RPQ and  $|\mathcal{G}|$  is the size of the graph database. We showed that it is possible to evaluate such UC2RPQs in time  $O(g(|\Phi|)|\mathcal{G}|^{k+1})$ , with  $g$  a singly exponential function. We did this by encoding the evaluation problem into a Datalog program of singly exponential size and width  $k + 1$ .

Besides the improvement in worst-case running time, the Datalog encoding also opens the way for practical approaches to evaluating UC2RPQs leveraging the plethora of available Datalog engines.

As open questions, a major one is a full characterisation of fixed parameter tractability of UC2RPQs, in particular establishing lower bounds. Another question concerns the precise complexity of computing semantic treewidth: for now it is known [12] that it is EXPSPACE-hard and in 2EXPSPACE.

---

## References

- 1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
- 2 Foto Afrati and Francesca Toni. Chain queries expressible by linear datalog programs. *Databases and Logic Programming*, pages 49–58, dec 1997.
- 3 Renzo Angles and Claudio Gutiérrez. Survey of graph database models. *ACM Computing Surveys (CSUR)*, 40(1):1–39, feb 2008. doi:10.1145/1322432.1322433.
- 4 Pablo Barceló, Leonid Libkin, Anthony W. Lin, and Peter T. Wood. Expressive languages for path queries over graph-structured data. *ACM Trans. Database Syst.*, 37(4), 2012. doi:10.1145/2389241.2389250.
- 5 Pablo Barceló, Jorge Pérez, and Juan L. Reutter. Relative expressiveness of nested regular expressions. In *Proc. of the 6th Alberto Mendelzon International Workshop on Foundations of Data Management, June 27-30, 2012*, volume 866, pages 180–195. CEUR-WS.org, 2012. URL: <https://ceur-ws.org/Vol-866/paper13.pdf>.
- 6 Pablo Barceló, Miguel Romero, and Moshe Y. Vardi. Semantic acyclicity on graph databases. *SIAM Journal on Computing*, 45(4):1339–1376, 2016. doi:10.1137/15M1034714.
- 7 Pablo Barceló. Querying graph databases. In *Proc. of the 32nd Symposium on Principles of Database Systems*, pages 175–188. ACM, 2013. doi:10.1145/2463664.2465216.
- 8 Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Containment of conjunctive regular path queries with inverse. In *Proc. of the Seventh International Conference on Principles of Knowledge Representation and Reasoning, KR'00*, pages 176–185, 2000.
- 9 Chandra Chekuri and Anand Rajaraman. Conjunctive query containment revisited. *Theor. Comput. Sci.*, 239(2):211–229, 2000. doi:10.1016/S0304-3975(99)00220-0.
- 10 Hubie Chen, Georg Gottlob, Matthias Lanzinger, and Reinhard Pichler. Semantic width and the fixed-parameter tractability of constraint satisfaction problems. In Christian Bessiere, editor, *IJCAI*, pages 1726–1733, 2020. doi:10.24963/ijcai.2020/239.
- 11 Alin Deutsch, Nadime Francis, Alastair Green, Keith Hare, Bei Li, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Wim Martens, Jan Michels, Filip Murlak, Stefan Plantikow, Petra Selmer, Oskar van Rest, Hannes Voigt, Domagoj Vrgoc, Mingxi Wu, and Fred Zemke. Graph pattern matching in GQL and SQL/PGQ. In Zachary Ives, Angela Bonifati, and Amr El Abbadi, editors, *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*, pages 2246–2258. ACM, 2022. doi:10.1145/3514221.3526057.
- 12 Diego Figueira and Rémi Morvan. Approximation and Semantic Tree-width of Conjunctive Regular Path Queries. In *26th International Conference on Database Theory*, Ioannina, Greece, mar 2023. Secondary link: <https://www.morvan.xyz/papers/main-crpq-tw-icdt-v1.pdf>. doi:10.4230/LIPIcs.ICDT.2023.15.
- 13 Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andrés Taylor. Cypher: An evolving query language for property graphs. In *Proc. of the 2018 International Conference on Management of Data, SIGMOD '18*, pages 1433–1445, 2018. doi:10.1145/3183713.3190657.
- 14 Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions and tractable queries. *Journal of Computer and System Sciences*, 64(3):579–627, 2002. doi:10.1006/jcss.2001.1809.
- 15 Martin Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *J. ACM*, 54(1):1:1–1:24, 2007. doi:10.1145/1206035.1206036.

- 16 Martin Grohe and Dániel Marx. Constraint solving via fractional edge covers. *ACM Trans. Algorithms*, 11(1), aug 2014. doi:10.1145/2636918.
- 17 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.
- 18 Dániel Marx. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. In *STOC*, pages 735–744, 2010. doi:10.1145/1806689.1806790.
- 19 Alberto O. Mendelzon and Peter T. Wood. Finding regular simple paths in graph databases. *SIAM Journal on Computing*, 24(6):1235–1258, 1995. doi:10.1137/S009753979122370X.
- 20 Miguel Romero, Pablo Barceló, and Moshe Y. Vardi. The homomorphism problem for regular graph patterns. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12, 2017. doi:10.1109/LICS.2017.8005106.
- 21 Sherif Sakr, Angela Bonifati, Hannes Voigt, Alexandru Iosup, Khaled Ammar, Renzo Angles, Walid Aref, Marcelo Arenas, Maciej Besta, Peter A. Boncz, Khuzaima Daudjee, Emanuele Della Valle, Stefania Dumbrava, Olaf Hartig, Bernhard Haslhofer, Tim Hegeman, Jan Hidders, Katja Hose, Adriana Iamnitchi, Vasiliki Kalavri, Hugo Kapp, Wim Martens, M. Tamer Özsu, Eric Peukert, Stefan Plantikow, Mohamed Ragab, Matei R. Ripeanu, Semih Salihoglu, Christian Schulz, Petra Selmer, Juan F. Sequeda, Joshua Shinavier, Gábor Szárnyas, Riccardo Tommasini, Antonino Tumeo, Alexandru Uta, Ana Lucia Varbanescu, Hsiang-Yun Wu, Nikolay Yakovets, Da Yan, and Eiko Yoneki. The future is big graphs: A community view on graph processing systems. *Commun. ACM*, 64(9):62–71, 2021. doi:10.1145/3434642.
- 22 Michael Sipser. *Introduction to the Theory of Computation*. Course Technology Inc., third edition, 2013.
- 23 Oskar van Rest, Sungpack Hong, Jinha Kim, Xuming Meng, and Hassan Chafi. PGQL: A property graph query language. In *Proceedings of the Fourth International Workshop on Graph Data Management Experiences and Systems*, pages 1–6. ACM, 2016. doi:10.1145/2960414.2960421.
- 24 Mihalis Yannakakis. Algorithms for acyclic database schemes. In *VLDB*, pages 82–94, 1981.

# Join Sampling Under Acyclic Degree Constraints and (Cyclic) Subgraph Sampling

Ru Wang ✉

The Chinese University of Hong Kong, China

Yufei Tao ✉

The Chinese University of Hong Kong, China

---

## Abstract

---

Given a (natural) join with an acyclic set of degree constraints (the join itself does not need to be acyclic), we show how to draw a uniformly random sample from the join result in  $O(\text{polymat}/\max\{1, \text{OUT}\})$  expected time (assuming data complexity) after a preprocessing phase of  $O(\text{IN})$  expected time, where IN, OUT, and *polymat* are the join’s input size, output size, and polymatroid bound, respectively. This compares favorably with the state of the art (Deng et al. and Kim et al., both in PODS’23), which states that, in the absence of degree constraints, a uniformly random sample can be drawn in  $\tilde{O}(\text{AGM}/\max\{1, \text{OUT}\})$  expected time after a preprocessing phase of  $\tilde{O}(\text{IN})$  expected time, where *AGM* is the join’s AGM bound and  $\tilde{O}(\cdot)$  hides a  $\text{polylog}(\text{IN})$  factor. Our algorithm applies to every join supported by the solutions of Deng et al. and Kim et al. Furthermore, since the polymatroid bound is at most the AGM bound, our performance guarantees are never worse, but can be considerably better, than those of Deng et al. and Kim et al.

We then utilize our techniques to tackle *directed subgraph sampling*, a problem that has extensive database applications and bears close relevance to joins. Let  $G = (V, E)$  be a directed data graph where each vertex has an out-degree at most  $\lambda$ , and let  $P$  be a directed pattern graph with a constant number of vertices. The objective is to uniformly sample an occurrence of  $P$  in  $G$ . The problem can be modeled as join sampling with input size  $\text{IN} = \Theta(|E|)$  but, whenever  $P$  contains cycles, the converted join has *cyclic* degree constraints. We show that it is always possible to throw away certain degree constraints such that (i) the remaining constraints are acyclic and (ii) the new join has asymptotically the same polymatroid bound *polymat* as the old one. Combining this finding with our new join sampling solution yields an algorithm to sample from the original (cyclic) join (thereby yielding a uniformly random occurrence of  $P$ ) in  $O(\text{polymat}/\max\{1, \text{OUT}\})$  expected time after  $O(|E|)$  expected-time preprocessing, where OUT is the number of occurrences.

**2012 ACM Subject Classification** Theory of computation → Graph algorithms analysis; Information systems → Join algorithms

**Keywords and phrases** Join Sampling, Subgraph Sampling, Degree Constraints, Polymatroid Bounds

**Digital Object Identifier** 10.4230/LIPIcs.ICDT.2024.23

**Related Version** *Full Version*: <https://arxiv.org/abs/2312.12797>

**Funding** This work was supported in part by GRF projects 14207820, 14203421, and 14222822 from HKRGC.

## 1 Introduction

In relational database systems, (natural) joins are acknowledged as notably computation-intensive, with its cost surging drastically in response to expanding data volumes. In the current big data era, the imperative to circumvent excessive computation increasingly overshadows the requirement for complete join results. A myriad of applications, including machine learning algorithms, online analytical processing, and recommendation systems, can operate effectively with random samples. This situation has sparked research initiatives focused on devising techniques capable of producing samples from a join result significantly



© Ru Wang and Yufei Tao;

licensed under Creative Commons License CC-BY 4.0

27th International Conference on Database Theory (ICDT 2024).

Editors: Graham Cormode and Michael Shekelyan; Article No. 23; pp. 23:1–23:20

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

faster than executing the join in its entirety. In the realm of graph theory, the significance of join operations is mirrored in their intrinsic connections to *subgraph listing*, a classical problem that seeks to pinpoint all the occurrences of a pattern  $P$  (for instance, a directed 3-vertex cycle) within a data graph  $G$  (such as a social network where a directed edge symbolizes a “follow” relationship). Analogous to joins, subgraph listing demands a vast amount of computation time, which escalates rapidly with the sizes of  $G$  and  $P$ . Fortunately, many social network analyses do not require the full set of occurrences of  $P$ , but can function well with only samples from those occurrences. This has triggered the development of methods that can extract samples considerably faster than finding all the occurrences.

This paper will revisit *join sampling* and *subgraph sampling* under a unified “degree-constrained framework”. Next, we will first describe the framework formally in Section 1.1, review the previous results in Section 1.2, and then overview our results in Section 1.3.

## 1.1 Problem Definitions

**Join Sampling.** Let  $\mathbf{att}$  be a finite set, with each element called an *attribute*, and  $\mathbf{dom}$  be a countably infinite set, with each element called a *value*. For a non-empty set  $\mathcal{X} \subseteq \mathbf{att}$  of attributes, a *tuple* over  $\mathcal{X}$  is a function  $\mathbf{u} : \mathcal{X} \rightarrow \mathbf{dom}$ . For any non-empty subset  $\mathcal{Y} \subseteq \mathcal{X}$ , we define  $\mathbf{u}[\mathcal{Y}]$  – the *projection* of  $\mathbf{u}$  on  $\mathcal{Y}$  – as the tuple  $\mathbf{v}$  over  $\mathcal{Y}$  satisfying  $\mathbf{v}(Y) = \mathbf{u}(Y)$  for every attribute  $Y \in \mathcal{Y}$ .

A *relation*  $R$  is a set of tuples over the same set  $\mathcal{Z}$  of attributes; we refer to  $\mathcal{Z}$  as the *schema* of  $R$  and represent it as  $schema(R)$ . The *arity* of  $R$  is the size of  $schema(R)$ . For any subsets  $\mathcal{X}$  and  $\mathcal{Y}$  of  $schema(R)$  satisfying  $\mathcal{X} \subset \mathcal{Y}$  (note:  $\mathcal{X}$  is a *proper* subset of  $\mathcal{Y}$ ), define:

$$deg_{\mathcal{Y}|\mathcal{X}}(R) = \max_{\text{tuple } \mathbf{u} \text{ over } \mathcal{X}} \left| \left\{ \mathbf{v}[\mathcal{Y}] \mid \mathbf{v} \in R, \mathbf{v}[\mathcal{X}] = \mathbf{u} \right\} \right|. \quad (1)$$

For an intuitive explanation, imagine grouping the tuples of  $R$  by  $\mathcal{X}$  and counting, for each group, how many *distinct*  $\mathcal{Y}$ -projections are formed by the tuples therein. Then, the value  $deg_{\mathcal{Y}|\mathcal{X}}(R)$  corresponds to the maximum count of all groups. It is worth pointing out that, when  $\mathcal{X} = \emptyset$ , then  $deg_{\mathcal{Y}|\mathcal{X}}(R)$  is simply  $|\Pi_{\mathcal{Y}}(R)|$  where  $\Pi$  is the standard “projection” operator in relational algebra. If in addition  $\mathcal{Y} = schema(R)$ , then  $deg_{\mathcal{Y}|\mathcal{X}}(R)$  equals  $|R|$ .

We define a *join* as a set  $\mathcal{Q}$  of relations (some of which may have the same schema). Let  $schema(\mathcal{Q})$  be the union of the attributes of the relations in  $\mathcal{Q}$ , i.e.,  $schema(\mathcal{Q}) = \bigcup_{R \in \mathcal{Q}} schema(R)$ . Focusing on “data complexity”, we consider only joins where both  $\mathcal{Q}$  and  $schema(\mathcal{Q})$  have constant sizes. The result of  $\mathcal{Q}$  is a relation over  $schema(\mathcal{Q})$  formalized as:

$$join(\mathcal{Q}) = \{ \text{tuple } \mathbf{u} \text{ over } schema(\mathcal{Q}) \mid \forall R \in \mathcal{Q} : \mathbf{u}[schema(R)] \in R \}.$$

Define  $IN = \sum_{R \in \mathcal{Q}} |R|$  and  $OUT = |join(\mathcal{Q})|$ . We will refer to  $IN$  and  $OUT$  as the *input size* and *output size* of  $\mathcal{Q}$ , respectively.

A *join sampling* operation returns a tuple drawn uniformly at random from  $join(\mathcal{Q})$  or declares  $join(\mathcal{Q}) = \emptyset$ . All such operations must be mutually independent. The objective of the *join sampling problem* is to preprocess the input relations of  $\mathcal{Q}$  into an appropriate data structure that can be used to perform join-sampling operations repeatedly.

We study the problem in the scenario where  $\mathcal{Q}$  conforms to a set  $DC$  of degree constraints. Specifically, each *degree constraint* has the form  $(\mathcal{X}, \mathcal{Y}, N_{\mathcal{Y}|\mathcal{X}})$  where  $\mathcal{X}$  and  $\mathcal{Y}$  are subsets of  $schema(\mathcal{Q})$  satisfying  $\mathcal{X} \subset \mathcal{Y}$  and  $N_{\mathcal{Y}|\mathcal{X}} \geq 1$  is an integer. A relation  $R \in \mathcal{Q}$  is said to *guard* the constraint  $(\mathcal{X}, \mathcal{Y}, N_{\mathcal{Y}|\mathcal{X}})$  if

$$\mathcal{Y} \subseteq schema(R), \text{ and } deg_{\mathcal{Y}|\mathcal{X}}(R) \leq N_{\mathcal{Y}|\mathcal{X}}.$$



The join  $\mathcal{Q}$  is *consistent* with DC – written as  $\mathcal{Q} \models \text{DC}$  – if every degree constraint in DC is guarded by at least one relation in  $\mathcal{Q}$ . It is safe to assume that DC does not have two constraints  $(\mathcal{X}, \mathcal{Y}, N_{\mathcal{Y}|\mathcal{X}})$  and  $(\mathcal{X}', \mathcal{Y}', N_{\mathcal{Y}'|\mathcal{X}'})$  with  $\mathcal{X} = \mathcal{X}'$  and  $\mathcal{Y} = \mathcal{Y}'$ ; otherwise, assuming  $N_{\mathcal{Y}|\mathcal{X}} \leq N_{\mathcal{Y}'|\mathcal{X}'}$ , the constraint  $(\mathcal{X}', \mathcal{Y}', N_{\mathcal{Y}'|\mathcal{X}'})$  is redundant and can be removed from DC.

In this work, we concentrate on “acyclic” degree dependency. To formalize this notion, let us define a *constraint dependency graph*  $G_{\text{DC}}$  as follows. This is a directed graph whose vertex set is  $\text{schema}(\mathcal{Q})$  (i.e., each vertex of  $G_{\text{DC}}$  is an attribute in  $\text{schema}(\mathcal{Q})$ ). For each degree constraint  $(\mathcal{X}, \mathcal{Y}, N_{\mathcal{Y}|\mathcal{X}})$  such that  $\mathcal{X} \neq \emptyset$ , we add a (directed) edge  $(X, Y)$  to  $G_{\text{DC}}$  for every pair  $(X, Y) \in \mathcal{X} \times (\mathcal{Y} - \mathcal{X})$ . We say that the set DC is *acyclic* if  $G_{\text{DC}}$  is an acyclic graph; otherwise, DC is *cyclic*.

It is important to note that each relation  $R \in \mathcal{Q}$  implicitly defines a special degree constraint  $(\mathcal{X}, \mathcal{Y}, N_{\mathcal{Y}|\mathcal{X}})$  where  $\mathcal{X} = \emptyset$ ,  $\mathcal{Y} = \text{schema}(R)$ , and  $N_{\mathcal{Y}|\mathcal{X}} = |R|$ . Such a constraint – known as a *cardinality constraint* – is always assumed to be present in DC. As all cardinality constraints have  $\mathcal{X} = \emptyset$ , they do not affect the construction of  $G_{\text{DC}}$ . Consequently, if DC only contains cardinality constraints, then  $G_{\text{DC}}$  is empty and hence trivially acyclic. Moreover, readers should avoid the misconception that “an acyclic  $G_{\text{DC}}$  implies  $\mathcal{Q}$  being an acyclic join”; these two acyclicity notions are unrelated. While the definition of an acyclic join is not needed for our discussion, readers unfamiliar with this term may refer to [2, Chapter 6.4].

**Directed Graph Sampling.** We are given a *data graph*  $G = (V, E)$  and a *pattern graph*  $P = (V_P, E_P)$ , both being simple directed graphs. The pattern graph is weakly-connected<sup>1</sup> and has a constant number of vertices. A simple directed graph  $G_{\text{sub}} = (V_{\text{sub}}, E_{\text{sub}})$  is a *subgraph* of  $G$  if  $V_{\text{sub}} \subseteq V$  and  $E_{\text{sub}} \subseteq E$ . The subgraph  $G_{\text{sub}}$  is an *occurrence* of  $P$  if they are isomorphic, namely, there is a bijection  $f : V_{\text{sub}} \rightarrow V_P$  such that, for any distinct vertices  $u_1, u_2 \in V_{\text{sub}}$ , there is an edge  $(u_1, u_2) \in E_{\text{sub}}$  if and only if  $(f(u_1), f(u_2))$  is an edge in  $E_P$ . We will refer to  $f$  as a *isomorphism bijection* between  $P$  and  $G_{\text{sub}}$ .

A *subgraph sampling* operation returns an occurrence of  $P$  in  $G$  uniformly at random or declares the absence of any occurrence. All such operations need to be mutually independent. The objective of the *subgraph sampling problem* is to preprocess  $G$  into a data structure that can support every subgraph-sampling operation efficiently. We will study the problem under a degree constraint: every vertex in  $G$  has an out-degree at most  $\lambda$ .

**Math Conventions.** For an integer  $x \geq 1$ , the notation  $[x]$  denotes the set  $\{1, 2, \dots, x\}$ ; as a special case,  $[0]$  represents the empty set. Every logarithm  $\log(\cdot)$  has base 2, and function  $\exp_2(x)$  is defined to be  $2^x$ . We use double curly braces to represent multi-sets, e.g.,  $\{\{1, 1, 1, 2, 2, 3\}\}$  is a multi-set with 6 elements.

## 1.2 Related Work

**Join Computation.** Any algorithm correctly answering a join query  $\mathcal{Q}$  must incur  $\Omega(\text{OUT})$  time just to output the OUT tuples in  $\text{join}(\mathcal{Q})$ . Hence, finding the greatest possible value of OUT is an imperative step towards unraveling the time complexity of join evaluation. A classical result in this regard is the *AGM bound* [6]. To describe this bound, let us define the *schema graph* of  $\mathcal{Q}$  as a multi-hypergraph  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$  where

$$\mathcal{V} = \text{schema}(\mathcal{Q}), \text{ and } \mathcal{E} = \{\{\text{schema}(R) \mid R \in \mathcal{Q}\}\}. \quad (2)$$

<sup>1</sup> Namely, if we ignore the edge directions, then  $P$  becomes a connected undirected graph.

Note that  $\mathcal{E}$  is a multi-set because the relations in  $\mathcal{Q}$  may have identical schemas. A *fractional edge cover* of  $\mathcal{G}$  is a function  $w : \mathcal{E} \rightarrow [0, 1]$  such that, for any  $X \in \mathcal{V}$ , it holds that  $\sum_{F \in \mathcal{E}: X \in F} w(F) \geq 1$  (namely, the total weight assigned to the hyperedges covering  $X$  is at least 1). Atserias, Grohe, and Marx [6] showed that, given any fractional edge cover, it always holds that  $\text{OUT} \leq \prod_{F \in \mathcal{E}} |R_F|^{w(F)}$ , where  $R_F$  is the relation in  $\mathcal{Q}$  whose schema corresponds to the hyperedge  $F$ . The AGM bound is defined as  $\text{AGM}(\mathcal{Q}) = \min_w \prod_{F \in \mathcal{E}} |R_F|^{w(F)}$ .

The AGM bound is tight: given any hypergraph  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$  and any set of positive integers  $\{N_F \mid F \in \mathcal{E}\}$ , there is always a join  $\mathcal{Q}$  such that  $\mathcal{Q}$  has  $\mathcal{G}$  as the schema graph,  $|R_F| = N_F$  for each  $F \in \mathcal{E}$ , and the output size  $\text{OUT}$  is  $\Theta(\text{AGM}(\mathcal{Q}))$ . This has motivated the development of algorithms [5, 13, 20, 22, 24, 27, 30–33, 36] that can compute  $\text{join}(\mathcal{Q})$  in  $\tilde{O}(\text{AGM}(\mathcal{Q}))$  time – where  $\tilde{O}(\cdot)$  hides a factor polylogarithmic to the input size  $\text{IN}$  of  $\mathcal{Q}$  – and therefore are worst-case optimal up to an  $\tilde{O}(1)$  factor.

However, the tightness of the AGM bound relies on the assumption that all the degree constraints on  $\mathcal{Q}$  are purely cardinality constraints. In reality, general degree constraints are prevalent, and their inclusion could dramatically decrease the maximum output size  $\text{OUT}$ . This observation has sparked significant interest [12, 16, 20, 21, 23, 24, 29, 34] in establishing refined upper bounds on  $\text{OUT}$  tailored for more complex degree constraints. Most notably, Khamis et al. [24] proposed the *entropic bound*, which is applicable to any set  $\text{DC}$  of degree constraints and is tight in a strong sense (see Theorem 5.5 of [34]). Unfortunately, the entropic bound is difficult to compute because it requires solving a linear program (LP) involving infinitely many constraints (it remains an open problem whether the computation is decidable). Not coincidentally, no join algorithm is known to have a running time matching the entropic bound.

To circumvent the above issue, Khamis et al. [24] introduced the *polymatroid bound* as an alternative, which we represent as  $\text{polymat}(\text{DC})$  because this bound is fully decided by  $\text{DC}$  (i.e., any join  $\mathcal{Q} \models \text{DC}$  must satisfy  $\text{OUT} \leq \text{polymat}(\text{DC})$ ). Section 2 will discuss  $\text{polymat}(\text{DC})$  in detail; for now, it suffices to understand that (i) the polymatroid bound, although possibly looser than the entropic bound, never exceeds the AGM bound, and (ii)  $\text{polymat}(\text{DC})$  can be computed in  $O(1)$  time under data complexity. Khamis et al. [24] proposed an algorithm named PANDA that can evaluate an arbitrary join  $\mathcal{Q} \models \text{DC}$  in time  $\tilde{O}(\text{polymat}(\text{DC}))$ .

Interestingly, when  $\text{DC}$  is acyclic, the entropic bound is equivalent to the polymatroid bound [29]. In this scenario, Ngo [29] presented a simple algorithm to compute any join  $\mathcal{Q} \models \text{DC}$  in  $O(\text{polymat}(\text{DC}))$  time, after a preprocessing of  $O(\text{IN})$  expected time.

**Join Sampling.** For an acyclic join (not to be confused with a join having an acyclic set of degree constraints), it is possible to sample from the join result in constant time, after a preprocessing of  $O(\text{IN})$  expected time [38]. The problem becomes more complex when dealing with an arbitrary (cyclic) join  $\mathcal{Q}$ , with the latest advancements presented in two PODS’23 papers [13, 25]. Specifically, Kim et al. [25] described how to sample in  $\tilde{O}(\text{AGM}(\mathcal{Q})/\max\{1, \text{OUT}\})$  expected time, after a preprocessing of  $\tilde{O}(\text{IN})$  time. Deng et al. [13] achieved the same guarantees using different approaches, and offered a rationale explaining why the expected sample time  $O(\text{AGM}(\mathcal{Q})/\text{OUT})$  can no longer be significantly improved, even when  $0 < \text{OUT} \ll \text{AGM}(\mathcal{Q})$ , subject to commonly accepted conjectures. We refer readers to [3, 9, 10, 13, 25, 38] and the references therein for other results (now superseded) on join sampling.

**Subgraph Listing.** Let us start by clarifying the *fractional edge cover number*  $\rho^*(P)$  of a simple undirected pattern graph  $P = (V_P, E_P)$ . Given a fractional edge cover of  $P$  (i.e., function  $w : E_P \rightarrow [0, 1]$  such that, for any vertex  $X \in V_P$ , we have  $\sum_{F \in E_P: X \in F} w(F) \geq 1$ ),

define  $\sum_{F \in E_P} w(F)$  as the *total weight* of  $w$ . The value of  $\rho^*(P)$  is the smallest total weight of all fractional edge covers of  $P$ . Given a directed pattern graph  $P$ , we define its fractional edge cover number  $\rho^*(P)$  as the value  $\rho^*(P')$  of the corresponding undirected graph  $P'$  that is obtained from  $P$  by ignoring all the edge directions.

When  $P$  has a constant size, it is well-known [4, 6] that any data graph  $G = (V, E)$  can encompass  $O(|E|^{\rho^*(P)})$  occurrences of  $P$ . This upper bound is tight: for any integer  $m$ , there is a data graph  $G = (V, E)$  with  $|E| = m$  edges that has  $\Omega(m^{\rho^*(P)})$  occurrences of  $P$ . Thus, a subgraph listing algorithm is considered worst-case optimal if it finishes in  $\tilde{O}(|E|^{\rho^*(P)})$  time.

It is well-known that subgraph listing can be converted to a join  $\mathcal{Q}$  on binary relations (namely, relations of arity 2). The join  $\mathcal{Q}$  has an input size of  $\text{IN} = \Theta(|E|)$ , and its AGM bound is  $\text{AGM}(\mathcal{Q}) = \Theta(|E|^{\rho^*(P)})$ . All occurrences of  $P$  in  $G$  can be derived from  $\text{join}(\mathcal{Q})$  for free. Thus, any  $\tilde{O}(\text{AGM}(\mathcal{Q}))$ -time join algorithm is essentially worst-case optimal for subgraph listing.

Assuming  $P$  and  $G$  to be directed, Jayaraman et al. [18] presented interesting enhancement over the above transformation in the scenario where each vertex of  $G$  has an out-degree at most  $\lambda$ . The key lies in examining the polymatroid bound of the join  $\mathcal{Q}$  derived from subgraph listing. As will be explained in Section 4, this join  $\mathcal{Q}$  has a set DC of degree constraints whose constraint dependency graph  $G_{\text{DC}}$  coincides with  $P$ . Jayaraman et al. developed an algorithm that lists all occurrences of  $\mathcal{Q}$  in  $G$  in  $O(\text{polymat}(\text{DC}))$  time (after a preprocessing of  $O(\text{IN})$  expected time) and confirmed that this is worst-case optimal. Their findings are closely related to our work, and we will delve into them further when their specifics become crucial to our discussion.

There is a substantial body of literature on bounding the cost of subgraph listing using parameters distinct from those already mentioned. These studies typically concentrate on specific patterns (such as paths, cycles, and cliques) or particular graphs (for instance, those that are sparse under a suitable metric). We refer interested readers to [1, 7, 8, 11, 14, 17, 19, 26, 28, 35] and the references therein.

**Subgraph Sampling.** Fichtenberger, Gao, and Peng [15] described how to sample an occurrence of the pattern  $P$  in the data graph  $G$  in  $O(|E|^{\rho^*(P)} / \max\{1, \text{OUT}\})$  expected time, where OUT is the number of occurrences of  $P$  in  $G$ , after a preprocessing of  $O(|E|)$  expected time. In [13], Deng et al. clarified how to deploy an arbitrary join sampling algorithm to perform subgraph sampling; their approach ensures the same guarantees as in [15], barring an  $\tilde{O}(1)$  factor.

### 1.3 Our Results

For any join  $\mathcal{Q}$  with an acyclic set DC of degree constraints, we will demonstrate in Section 3 how to extract a uniformly random sample from  $\text{join}(\mathcal{Q})$  in  $O(\text{polymat}(\text{DC}) / \max\{1, \text{OUT}\})$  expected time, following an initial preprocessing of  $O(\text{IN})$  expected time. This performance is favorable when compared to the recent results of [13, 25] (reviewed in Section 1.2), which examined settings where DC consists only of cardinality constraints and is therefore trivially acyclic. As  $\text{polymat}(\text{DC})$  is at most but can be substantially lower than  $\text{AGM}(\mathcal{Q})$ , our guarantees are never worse, but can be considerably better, than those in [13, 25].

What if DC is cyclic? An idea, proposed in [29], is to discard enough constraints to make the remaining set  $\text{DC}'$  of constraints acyclic (while ensuring  $\mathcal{Q} \models \text{DC}'$ ). Our algorithm can then be applied to draw a sample in  $O(\text{polymat}(\text{DC}') / \max\{1, \text{OUT}\})$  time. However, this can be unsatisfactory because  $\text{polymat}(\text{DC}')$  can potentially be much larger than  $\text{polymat}(\text{DC})$ .

Our next contribution is to prove that, interestingly, the issue does not affect subgraph listing/sampling. Consider first directed subgraph listing, defined by a pattern graph  $P$  and a data graph  $G$  where every vertex has an out-degree at most  $\lambda$ . This problem can be converted to a join  $Q$  on binary relations, which is associated with a set  $DC$  of degree constraints such that the constraint dependency graph  $G_{DC}$  is exactly  $P$ . Consequently, whenever  $P$  contains a cycle, so does  $G_{DC}$ , making  $DC$  cyclic. Nevertheless, we will demonstrate in Section 4 the existence of an acyclic set  $DC' \subset DC$  ensuring  $Q \models DC'$  and  $\text{polymat}(DC) = \Theta(\text{polymat}(DC'))$ . This “magical”  $DC'$  has an immediate implication: Ngo’s join algorithm in [29], when applied to  $Q$  and  $DC'$  directly, already solves directed subgraph listing optimally in  $O(\text{polymat}(DC')) = O(\text{polymat}(DC))$  time. This dramatically simplifies – in terms of both procedure and analysis – an algorithm of Jayaraman et al. [18] (for directed subgraph listing, reviewed in Section 1.2) that has the same guarantees.

The same elegance extends to directed subgraph sampling: by applying our new join sampling algorithm to  $Q$  and the “magical”  $DC'$ , we can sample an occurrence of  $P$  in  $G$  using  $O(\text{polymat}(DC)/\max\{1, \text{OUT}\})$  expected time, after a preprocessing of  $O(|E|)$  expected time. As  $\text{polymat}(DC)$  never exceeds but can be much lower than  $AGM(Q) = \Theta(|E|^{\rho^*(P)})$ , our result compares favorably with the state of the art [13, 15, 25] reviewed in Section 1.2.

In the full version of this paper [37], we will prove similar results for *undirected subgraph sampling* and demonstrate how our techniques can be significantly simplified in that scenario. By virtue of the power of sampling, our findings have further implications on other fundamental problems including output-size estimation, output permutation, and small-delay enumeration, as discussed in Section 5.

## 2 Preliminaries

**Set Functions, Polymatroid Bounds, and Modular Bounds.** Suppose that  $\mathcal{S}$  is a finite set. We refer to a function  $h : 2^{\mathcal{S}} \rightarrow \mathbb{R}_{\geq 0}$  as a *set function over  $\mathcal{S}$* , where  $\mathbb{R}_{\geq 0}$  is the set of non-negative real values. Such a function  $h$  is said to be

- *zero-grounded* if  $h(\emptyset) = 0$ ;
- *monotone* if  $h(\mathcal{X}) \leq h(\mathcal{Y})$  for all  $\mathcal{X}, \mathcal{Y}$  satisfying  $\mathcal{X} \subseteq \mathcal{Y} \subseteq \mathcal{S}$ ;
- *modular* if  $h(\mathcal{X}) = \sum_{A \in \mathcal{X}} h(\{A\})$  holds for any  $\mathcal{X} \subseteq \mathcal{S}$ ;
- *submodular* if  $h(\mathcal{X} \cup \mathcal{Y}) + h(\mathcal{X} \cap \mathcal{Y}) \leq h(\mathcal{X}) + h(\mathcal{Y})$  holds for any  $\mathcal{X}, \mathcal{Y} \subseteq \mathcal{S}$ .

Define:

$\mathbf{M}_{\mathcal{S}}$  = the set of modular set functions over  $\mathcal{S}$

$\Gamma_{\mathcal{S}}$  = the set of set functions over  $\mathcal{S}$  that are zero-grounded, monotone, submodular

Note that every modular function must be zero-grounded and monotone. Clearly,  $\mathbf{M}_{\mathcal{S}} \subseteq \Gamma_{\mathcal{S}}$ .

Consider  $\mathcal{C}$  to be a set of triples, each having the form  $(\mathcal{X}, \mathcal{Y}, N_{\mathcal{Y}|\mathcal{X}})$  where  $\mathcal{X} \subset \mathcal{Y} \subseteq \mathcal{S}$  and  $N_{\mathcal{Y}|\mathcal{X}} \geq 1$  is an integer. We will refer to  $\mathcal{C}$  as a *rule collection over  $\mathcal{S}$*  and to each triple therein as a *rule*. Intuitively, the presence of a rule collection is to instruct us to focus only on certain restricted set functions. Formally, these are the set functions in:

$$\mathcal{H}_{\mathcal{C}} = \{\text{set function } h \text{ over } \mathcal{S} \mid h(\mathcal{Y}) - h(\mathcal{X}) \leq \log N_{\mathcal{Y}|\mathcal{X}}, \forall (\mathcal{X}, \mathcal{Y}, N_{\mathcal{Y}|\mathcal{X}}) \in \mathcal{C}\}. \quad (3)$$

The *polymatroid bound* of  $\mathcal{C}$  can now be defined as

$$\text{polymat}(\mathcal{C}) = \exp_2 \left( \max_{h \in \Gamma_{\mathcal{S}} \cap \mathcal{H}_{\mathcal{C}}} h(\mathcal{S}) \right). \quad (4)$$

Recall that  $\exp_2(x) = 2^x$ . Similarly, the *modular bound* of  $\mathcal{C}$  is defined as

$$\text{modular}(\mathcal{C}) = \exp_2 \left( \max_{h \in \mathbf{M}_{\mathcal{S}} \cap \mathcal{H}_{\mathcal{C}}} h(\mathcal{S}) \right). \quad (5)$$

**Join Output Size Bounds.** Let us fix a join  $\mathcal{Q}$  whose schema graph is  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ . Suppose that  $\mathcal{Q}$  is consistent with a set DC of degree constraints, i.e.,  $\mathcal{Q} \models \text{DC}$ . As explained in Section 1.1, we follow the convention that each relation of  $\mathcal{Q}$  implicitly inserts a cardinality constraint (i.e., a special degree constraint) to DC. Note that the set DC is merely a rule collection over  $\mathcal{V}$ . The following lemma was established by Khamis et al. [24]:

► **Lemma 1** ([24]). *The output size OUT of  $\mathcal{Q}$  is at most  $\text{polymat}(\text{DC})$ , i.e., the polymatroid bound of DC (as defined in (4)).*

How about  $\text{modular}(\text{DC})$ , i.e., the modular bound of  $\mathcal{V}$ ? As  $\text{M}_{\mathcal{V}} \subseteq \Gamma_{\mathcal{V}}$ , we have  $\text{modular}(\text{DC}) \leq \text{polymat}(\text{DC})$  and the inequality can be strict in general. However, an exception arises when DC is acyclic, as proved in [29]:

► **Lemma 2** ([29]). *When DC is acyclic, it always holds that  $\text{modular}(\text{DC}) = \text{polymat}(\text{DC})$ , namely,  $\max_{h \in \Gamma_{\mathcal{V}} \cap \mathcal{H}_{\text{DC}}} h(\mathcal{V}) = \max_{h \in \text{M}_{\mathcal{V}} \cap \mathcal{H}_{\text{DC}}} h(\mathcal{V})$ .*

As a corollary, when DC is acyclic, the value of  $\text{modular}(\text{DC})$  always serves as an upper bound of OUT. In our technical development, we will need to analyze the set functions  $h^* \in \Gamma_{\mathcal{V}}$  that realize the polymatroid bound, i.e.,  $h^*(\mathcal{V}) = \max_{h \in \Gamma_{\mathcal{V}} \cap \mathcal{H}_{\text{DC}}} h(\mathcal{V})$ . A crucial advantage provided by Lemma 2 is that we can instead scrutinize those set functions  $h^* \in \text{M}_{\mathcal{V}}$  realizing the *modular* bound, i.e.,  $h^*(\mathcal{V}) = \max_{h \in \text{M}_{\mathcal{V}} \cap \mathcal{H}_{\text{DC}}} h(\mathcal{V})$ . Compared to their submodular counterparts, modular set functions exhibit more regularity because every  $h \in \text{M}_{\mathcal{V}}$  is fully determined by its value  $h(\{A\})$  on each *individual* attribute  $A \in \mathcal{V}$ . In particular, for any  $h \in \text{M}_{\mathcal{V}} \cap \mathcal{H}_{\text{DC}}$ , it holds true that  $h(\mathcal{Y}) - h(\mathcal{X}) = \sum_{A \in \mathcal{Y} - \mathcal{X}} h(A)$  for any  $\mathcal{X} \subset \mathcal{Y} \subseteq \mathcal{V}$ . If we associate each  $A \in \mathcal{V}$  with a variable  $\nu_A$ , then  $\max_{h \in \text{M}_{\mathcal{V}} \cap \mathcal{H}_{\text{DC}}} h(\mathcal{V})$  – hence, also  $\max_{h \in \Gamma_{\mathcal{V}} \cap \mathcal{H}_{\text{DC}}} h(\mathcal{V})$  – is precisely the optimal value of the following LP:

$$\begin{aligned} \text{modular LP} \quad & \max \sum_{A \in \mathcal{V}} \nu_A \text{ subject to} \\ & \sum_{A \in \mathcal{Y} - \mathcal{X}} \nu_A \leq \log N_{\mathcal{Y}|\mathcal{X}} \quad \forall (\mathcal{X}, \mathcal{Y}, N_{\mathcal{Y}|\mathcal{X}}) \in \text{DC} \\ & \nu_A \geq 0 \quad \forall A \in \mathcal{V} \end{aligned}$$

We will also need to work with the LP's dual. Specifically, if we associate a variable  $\delta_{\mathcal{Y}|\mathcal{X}}$  for every degree constraint  $(\mathcal{X}, \mathcal{Y}, N_{\mathcal{Y}|\mathcal{X}}) \in \text{DC}$ , then the dual LP is:

$$\begin{aligned} \text{dual modular LP} \quad & \min \sum_{(\mathcal{X}, \mathcal{Y}, N_{\mathcal{Y}|\mathcal{X}}) \in \text{DC}} \delta_{\mathcal{Y}|\mathcal{X}} \cdot \log N_{\mathcal{Y}|\mathcal{X}} \text{ subject to} \\ & \sum_{\substack{(\mathcal{X}, \mathcal{Y}, N_{\mathcal{Y}|\mathcal{X}}) \in \text{DC} \\ A \in \mathcal{Y} - \mathcal{X}}} \delta_{\mathcal{Y}|\mathcal{X}} \geq 1 \quad \forall A \in \mathcal{V} \\ & \delta_{\mathcal{Y}|\mathcal{X}} \geq 0 \quad \forall (\mathcal{X}, \mathcal{Y}, N_{\mathcal{Y}|\mathcal{X}}) \in \text{DC} \end{aligned}$$

### 3 Join Sampling under Acyclic Degree Dependency

This section serves as a proof of our first main result:

► **Theorem 3.** *For any join  $\mathcal{Q}$  consistent with an acyclic set DC of degree constraints, we can build in  $O(\text{IN})$  expected time a data structure that supports each join sampling operation in  $O(\text{polymat}(\text{DC}) / \max\{1, \text{OUT}\})$  expected time, where IN and OUT are the input and out sizes of  $\mathcal{Q}$ , respectively, and  $\text{polymat}(\text{DC})$  is the polymatroid bound of DC.*

**Basic Definitions.** Let  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$  be the schema graph of  $\mathcal{Q}$ , and  $G_{\text{DC}}$  be the constraint dependency graph determined by DC. For each hyperedge  $F \in \mathcal{E}$ , we denote by  $R_F$  the relation whose schema corresponds to  $F$ . Recall that every constraint  $(\mathcal{X}, \mathcal{Y}, N_{\mathcal{Y}|\mathcal{X}}) \in \text{DC}$  is guarded by at least one relation in  $\mathcal{Q}$ . Among them, we arbitrarily designate one relation as the constraint's *main guard*, whose schema is represented as  $F(\mathcal{X}, \mathcal{Y})$  (the main guard can then be conveniently identified as  $R_{F(\mathcal{X}, \mathcal{Y})}$ ).

Set  $k = |\mathcal{V}|$ . As  $G_{\text{DC}}$  is a DAG (acyclic directed graph), we can order its  $k$  vertices (i.e., attributes) into a topological order:  $A_1, A_2, \dots, A_k$ . For each  $i \in [k]$ , define  $\mathcal{V}_i = \{A_1, A_2, \dots, A_i\}$ ; specially, define  $\mathcal{V}_0 = \emptyset$ . For any  $i \in [k]$ , define

$$\text{DC}(A_i) = \{(\mathcal{X}, \mathcal{Y}, N_{\mathcal{Y}|\mathcal{X}}) \in \text{DC} \mid A_i \in \mathcal{Y} - \mathcal{X}\} \quad (6)$$

Fix an arbitrary  $i \in [k]$  and an arbitrary constraint  $(\mathcal{X}, \mathcal{Y}, N_{\mathcal{Y}|\mathcal{X}}) \in \text{DC}(A_i)$ . Given a tuple  $\mathbf{w}$  over  $\mathcal{V}_{i-1}$  (note: if  $i = 1$ , then  $\mathcal{V}_{i-1} = \emptyset$  and  $\mathbf{w}$  is a null tuple) and a value  $a \in \text{dom}$ , we define a “relative degree” for  $a$  as:

$$\text{reldeg}_{i, \mathcal{X}, \mathcal{Y}}(\mathbf{w}, a) = \frac{|\sigma_{A_i=a}(\Pi_{\mathcal{Y}}(R_{F(\mathcal{X}, \mathcal{Y})} \times \mathbf{w}))|}{|\Pi_{\mathcal{Y}}(R_{F(\mathcal{X}, \mathcal{Y})} \times \mathbf{w})|} \quad (7)$$

where  $\sigma$  and  $\times$  are the standard selection and semi-join operators in relational algebra, respectively. To understand the intuition behind  $\text{reldeg}_{i, \mathcal{X}, \mathcal{Y}}(\mathbf{w}, a)$ , imagine drawing a tuple  $\mathbf{u}$  from  $\Pi_{\mathcal{Y}}(R_{F(\mathcal{X}, \mathcal{Y})} \times \mathbf{w})$  uniformly at random; then  $\text{reldeg}_{i, \mathcal{X}, \mathcal{Y}}(\mathbf{w}, a)$  is the probability to see  $\mathbf{u}(A_i) = a$ . Given a tuple  $\mathbf{w}$  over  $\mathcal{V}_{i-1}$  and a value  $a \in \text{dom}$ , define

$$\text{reldeg}_i^*(\mathbf{w}, a) = \max_{(\mathcal{X}, \mathcal{Y}, N_{\mathcal{Y}|\mathcal{X}}) \in \text{DC}(A_i)} \text{reldeg}_{i, \mathcal{X}, \mathcal{Y}}(\mathbf{w}, a) \quad (8)$$

$$\text{constraint}_i^*(\mathbf{w}, a) = \arg \max_{(\mathcal{X}, \mathcal{Y}, N_{\mathcal{Y}|\mathcal{X}}) \in \text{DC}(A_i)} \text{reldeg}_{i, \mathcal{X}, \mathcal{Y}}(\mathbf{w}, a). \quad (9)$$

Specifically,  $\text{constraint}_i^*(\mathbf{w}, a)$  returns the constraint  $(\mathcal{X}, \mathcal{Y}, N_{\mathcal{Y}|\mathcal{X}}) \in \text{DC}(A_i)$  satisfying the condition  $\text{reldeg}_{i, \mathcal{X}, \mathcal{Y}}(\mathbf{w}, a) = \text{reldeg}_i^*(\mathbf{w}, a)$ . If more than one constraint meets this condition, define  $\text{constraint}_i^*(\mathbf{w}, a)$  to be an arbitrary one among those constraints.

Henceforth, we will fix an arbitrary optimal solution  $\{\delta_{\mathcal{Y}|\mathcal{X}} \mid (\mathcal{X}, \mathcal{Y}, N_{\mathcal{Y}|\mathcal{X}}) \in \text{DC}\}$  to the dual modular LP in Section 2. Thus:

$$\begin{aligned} \prod_{(\mathcal{X}, \mathcal{Y}, N_{\mathcal{Y}|\mathcal{X}}) \in \text{DC}} N_{\mathcal{Y}|\mathcal{X}}^{\delta_{\mathcal{Y}|\mathcal{X}}} &= \exp_2 \left( \sum_{(\mathcal{X}, \mathcal{Y}, N_{\mathcal{Y}|\mathcal{X}}) \in \text{DC}} \delta_{\mathcal{Y}|\mathcal{X}} \cdot \log N_{\mathcal{Y}|\mathcal{X}} \right) = \exp_2 \left( \max_{h \in \mathcal{M}_{\mathcal{V}} \cap \mathcal{H}_{\text{DC}}} h(\mathcal{V}) \right) \\ &\text{(by (5))} = \text{modular}(\text{DC}) \\ &\text{(by Lemma 2)} = \text{polymat}(\text{DC}). \end{aligned} \quad (10)$$

Finally, for any  $i \in [0, k]$  and any tuple  $\mathbf{w}$  over  $\mathcal{V}_i$ , define:

$$B_i(\mathbf{w}) = \prod_{(\mathcal{X}, \mathcal{Y}, N_{\mathcal{Y}|\mathcal{X}}) \in \text{DC}} (\text{deg}_{\mathcal{Y}|\mathcal{X}}(R_{F(\mathcal{X}, \mathcal{Y})} \times \mathbf{w}))^{\delta_{\mathcal{Y}|\mathcal{X}}}. \quad (11)$$

Two observations will be useful later:

- If  $i = 0$ , then  $\mathbf{w}$  is a null tuple and  $B_0(\text{null}) = \prod_{(\mathcal{X}, \mathcal{Y}, N_{\mathcal{Y}|\mathcal{X}}) \in \text{DC}} (\text{deg}_{\mathcal{Y}|\mathcal{X}}(R_{F(\mathcal{X}, \mathcal{Y})}))^{\delta_{\mathcal{Y}|\mathcal{X}}}$ , which is at most  $\prod_{(\mathcal{X}, \mathcal{Y}, N_{\mathcal{Y}|\mathcal{X}}) \in \text{DC}} N_{\mathcal{Y}|\mathcal{X}}^{\delta_{\mathcal{Y}|\mathcal{X}}} = \text{polymat}(\text{DC})$ .
- If  $i = k$  and  $\mathbf{w} \in \text{join}(\mathcal{Q})$ , then  $R_{F(\mathcal{X}, \mathcal{Y})} \times \mathbf{w}$  contains exactly one tuple for any  $(\mathcal{X}, \mathcal{Y}, N_{\mathcal{Y}|\mathcal{X}}) \in \text{DC}$  and thus  $B_k(\mathbf{w}) = 1$ .

■ **Algorithm 1** Our sampling algorithm.

ADC-sample

0.  $A_1, A_2, \dots, A_k \leftarrow$  a topological order of  $G_{DC}$
1.  $\mathbf{w}_0 \leftarrow$  a null tuple
2. **for**  $i = 1$  to  $k$  **do**
3.   pick a constraint  $(\mathcal{X}^\circ, \mathcal{Y}^\circ, N_{\mathcal{Y}^\circ|\mathcal{X}^\circ})$  uniformly at random from  $DC(A_i)$
4.    $\mathbf{u}^\circ \leftarrow$  a tuple chosen uniformly at random from  $\Pi_{\mathcal{Y}^\circ}(R_{F(\mathcal{X}^\circ, \mathcal{Y}^\circ)} \times \mathbf{w}_{i-1})$   
/\* note: if  $i = 1$ , then  $R_{F(\mathcal{X}^\circ, \mathcal{Y}^\circ)} \times \mathbf{w}_{i-1} = R_{F(\mathcal{X}^\circ, \mathcal{Y}^\circ)}$  \*/
5.    $a_i \leftarrow \mathbf{u}^\circ(A_i)$
6.   **if**  $(\mathcal{X}^\circ, \mathcal{Y}^\circ, N_{\mathcal{Y}^\circ|\mathcal{X}^\circ}) \neq \text{constraint}_i^*(\mathbf{w}_{i-1}, a_i)$  **then** declare **failure**
7.    $\mathbf{w}_i \leftarrow$  the tuple over  $\mathcal{V}_i$  formed by concatenating  $\mathbf{w}_{i-1}$  with  $a_i$
8.   declare **failure** with probability  $1 - p_{\text{pass}}(i, \mathbf{w}_{i-1}, \mathbf{w}_i)$ , where  $p_{\text{pass}}$  is given in (12)
9. **if**  $\mathbf{w}_k[F] \in R_F$  for  $\forall F \in \mathcal{E}$  **then**               /\* that is,  $\mathbf{w}_k \in \text{join}(\mathcal{Q})$  \*/
10. **return**  $\mathbf{w}_k$

**Algorithm.** Our sampling algorithm, named ADC-sample, is presented in Algorithm 1. At a high level, it processes one attribute at a time according to the topological order  $A_1, A_2, \dots, A_k$ . The for-loop in Lines 2–9 finds a value  $a_i$  for attribute  $A_i$  ( $i \in [k]$ ). The algorithm may fail to return anything, but when it *succeeds* (i.e., reaching Line 10), the values  $a_1, a_2, \dots, a_k$  will make a uniformly random tuple from  $\text{join}(\mathcal{Q})$ .

Next, we explain the details of the for-loop. The loop starts with values  $a_1, a_2, \dots, a_{i-1}$  already stored in a tuple  $\mathbf{w}_{i-1}$  (i.e.,  $\mathbf{w}_{i-1}(A_j) = a_j$  for all  $j \in [i-1]$ ). Line 3 randomly chooses a degree constraint  $(\mathcal{X}^\circ, \mathcal{Y}^\circ, N_{\mathcal{Y}^\circ|\mathcal{X}^\circ})$  from  $DC(A_i)$ ; see (6). Conceptually, next we identify the main guard  $R_{F(\mathcal{X}^\circ, \mathcal{Y}^\circ)}$  of this constraint, semi-join the relation with  $\mathbf{w}_{i-1}$ , and project the semi-join result on  $\mathcal{Y}^\circ$  to obtain  $\Pi_{\mathcal{Y}^\circ}(R_{F(\mathcal{X}^\circ, \mathcal{Y}^\circ)} \times \mathbf{w}_{i-1})$ . Then, Line 4 randomly chooses a tuple  $\mathbf{u}^\circ$  from  $\Pi_{\mathcal{Y}^\circ}(R_{F(\mathcal{X}^\circ, \mathcal{Y}^\circ)} \times \mathbf{w}_{i-1})$  and Line 5 takes  $\mathbf{u}^\circ(A_i)$  as the value of  $a_i$  (note:  $A_i \in \mathcal{Y}^\circ - \mathcal{X}^\circ \subseteq \mathcal{Y}^\circ$ ). Physically, however, we do not compute  $\Pi_{\mathcal{Y}^\circ}(R_{F(\mathcal{X}^\circ, \mathcal{Y}^\circ)} \times \mathbf{w}_{i-1})$  during the sample process. Instead, with proper preprocessing (discussed later), we can acquire the value  $a_i$  in  $O(1)$  time. Continuing, Line 6 may declare failure and terminate ADC-sample, but if we get past this line,  $(\mathcal{X}^\circ, \mathcal{Y}^\circ, N_{\mathcal{Y}^\circ|\mathcal{X}^\circ})$  must be exactly  $\text{constraint}_i^*(\mathbf{w}_{i-1}, a_i)$ ; see (9). As clarified later, the check at Line 6 can be performed in  $O(1)$  time. We now form a tuple  $\mathbf{w}_i$  that takes value  $a_j$  on attribute  $A_j$  for each  $j \in [i]$  (Line 7). Line 8 allows us to pass with probability

$$p_{\text{pass}}(i, \mathbf{w}_{i-1}, \mathbf{w}_i) = \frac{B_i(\mathbf{w}_i)}{B_{i-1}(\mathbf{w}_{i-1})} \cdot \frac{1}{\text{reldeg}_i^*(\mathbf{w}_{i-1}, \mathbf{w}_i(A_i))} \quad (12)$$

or otherwise terminate the algorithm by declaring failure. As proved later,  $p_{\text{pass}}(i, \mathbf{w}_{i-1}, \mathbf{w}_i)$  cannot exceed 1 (Lemma 4); moreover, this value can be computed in  $O(1)$  time. The overall execution time of ADC-sample is constant.

**Analysis.** Next we prove that the value in (12) serves as a legal probability value.

► **Lemma 4.** For every  $i \in [k]$ , we have  $p_{\text{pass}}(i, \mathbf{w}_{i-1}, \mathbf{w}_i) \leq 1$ .

**Proof.** Consider an arbitrary constraint  $(\mathcal{X}, \mathcal{Y}, N_{\mathcal{Y}|\mathcal{X}}) \in DC(A_i)$ . Recall that ADC-sample processes the attributes by the topological order  $A_1, \dots, A_k$ . In the constrained dependency graph  $G_{DC}$ , every attribute of  $\mathcal{X}$  has an out-going edge to  $A_i$ . Hence, all the attributes in  $\mathcal{X}$  must be processed prior to  $A_i$ . This implies that all the tuples in  $R_{F(\mathcal{X}, \mathcal{Y})} \times \mathbf{w}_{i-1}$  must have the same projection on  $\mathcal{X}$ . Therefore,  $\text{deg}_{\mathcal{Y}|\mathcal{X}}(R_{F(\mathcal{X}, \mathcal{Y})} \times \mathbf{w}_{i-1})$  equals  $|\Pi_{\mathcal{Y}}(R_{F(\mathcal{X}, \mathcal{Y})} \times \mathbf{w}_{i-1})|$ . By the same reasoning,  $\text{deg}_{\mathcal{Y}|\mathcal{X}}(R_{F(\mathcal{X}, \mathcal{Y})} \times \mathbf{w}_i)$  equals  $|\Pi_{\mathcal{Y}}(R_{F(\mathcal{X}, \mathcal{Y})} \times \mathbf{w}_i)|$ . We thus have:



$$\begin{aligned}
 \frac{\deg_{\mathcal{Y}|\mathcal{X}}(R_{F(\mathcal{X},\mathcal{Y})} \times \mathbf{w}_i)}{\deg_{\mathcal{Y}|\mathcal{X}}(R_{F(\mathcal{X},\mathcal{Y})} \times \mathbf{w}_{i-1})} &= \frac{|\Pi_{\mathcal{Y}}(R_{F(\mathcal{X},\mathcal{Y})} \times \mathbf{w}_i)|}{|\Pi_{\mathcal{Y}}(R_{F(\mathcal{X},\mathcal{Y})} \times \mathbf{w}_{i-1})|} \\
 &= \frac{|\sigma_{A_i=a_i}(\Pi_{\mathcal{Y}}(R_{F(\mathcal{X},\mathcal{Y})} \times \mathbf{w}_{i-1}))|}{|\Pi_{\mathcal{Y}}(R_{F(\mathcal{X},\mathcal{Y})} \times \mathbf{w}_{i-1})|} \\
 &= \text{reldeg}_{i,\mathcal{X},\mathcal{Y}}(\mathbf{w}_{i-1}, a_i) \\
 &\leq \text{reldeg}_i^*(\mathbf{w}_{i-1}, a_i). \tag{13}
 \end{aligned}$$

On the other hand, for any constraint  $(\mathcal{X}, \mathcal{Y}, N_{\mathcal{Y}|\mathcal{X}}) \notin \text{DC}(A_i)$ , it trivially holds that

$$\deg_{\mathcal{Y}|\mathcal{X}}(R_{F(\mathcal{X},\mathcal{Y})} \times \mathbf{w}_i) \leq \deg_{\mathcal{Y}|\mathcal{X}}(R_{F(\mathcal{X},\mathcal{Y})} \times \mathbf{w}_{i-1}) \tag{14}$$

because  $R_{F(\mathcal{X},\mathcal{Y})} \times \mathbf{w}_i$  is a subset of  $R_{F(\mathcal{X},\mathcal{Y})} \times \mathbf{w}_{i-1}$ .

We can now derive

$$\begin{aligned}
 p_{\text{pass}}(i, \mathbf{w}_{i-1}, \mathbf{w}_i) &= \frac{1}{\text{reldeg}_i^*(\mathbf{w}_{i-1}, a_i)} \prod_{(\mathcal{X}, \mathcal{Y}, N_{\mathcal{Y}|\mathcal{X}}) \in \text{DC}} \left( \frac{\deg_{\mathcal{Y}|\mathcal{X}}(R_{F(\mathcal{X},\mathcal{Y})} \times \mathbf{w}_i)}{\deg_{\mathcal{Y}|\mathcal{X}}(R_{F(\mathcal{X},\mathcal{Y})} \times \mathbf{w}_{i-1})} \right)^{\delta_{\mathcal{Y}|\mathcal{X}}} \\
 \text{(by (14))} &\leq \frac{1}{\text{reldeg}_i^*(\mathbf{w}_{i-1}, a_i)} \prod_{(\mathcal{X}, \mathcal{Y}, N_{\mathcal{Y}|\mathcal{X}}) \in \text{DC}(A_i)} \left( \frac{\deg_{\mathcal{Y}|\mathcal{X}}(R_{F(\mathcal{X},\mathcal{Y})} \times \mathbf{w}_i)}{\deg_{\mathcal{Y}|\mathcal{X}}(R_{F(\mathcal{X},\mathcal{Y})} \times \mathbf{w}_{i-1})} \right)^{\delta_{\mathcal{Y}|\mathcal{X}}} \\
 \text{(by (13))} &\leq \frac{1}{\text{reldeg}_i^*(\mathbf{w}_{i-1}, a_i)} \prod_{(\mathcal{X}, \mathcal{Y}, N_{\mathcal{Y}|\mathcal{X}}) \in \text{DC}(A_i)} \text{reldeg}_i^*(\mathbf{w}_{i-1}, a_i)^{\delta_{\mathcal{Y}|\mathcal{X}}} \\
 &= \text{reldeg}_i^*(\mathbf{w}_{i-1}, a_i)^{\left( \sum_{(\mathcal{X}, \mathcal{Y}, N_{\mathcal{Y}|\mathcal{X}}) \in \text{DC}(A_i)} \delta_{\mathcal{Y}|\mathcal{X}} \right) - 1} \leq 1.
 \end{aligned}$$

The last step used  $\sum_{(\mathcal{X}, \mathcal{Y}, N_{\mathcal{Y}|\mathcal{X}}) \in \text{DC}(A_i)} \delta_{\mathcal{Y}|\mathcal{X}} \geq 1$  guaranteed by the dual modular LP.  $\blacktriangleleft$

Next, we argue that every result tuple  $\mathbf{v} \in \text{join}(\mathcal{Q})$  is returned by ADC-sample with the same probability. For this purpose, let us define two random events for each  $i \in [k]$ :

- event **E1**( $i$ ):  $(\mathcal{X}^\circ, \mathcal{Y}^\circ, N_{\mathcal{Y}^\circ|\mathcal{X}^\circ}) = \text{constraint}_i^*(\mathbf{w}_{i-1}, \mathbf{v}(A_i))$  in the  $i$ -th loop of ADC-sample;
- event **E2**( $i$ ): Line 8 does not declare failure in the  $i$ -th loop of ADC-sample.

The probability for ADC-sample to return  $\mathbf{v}$  can be derived as follows.

$$\begin{aligned}
 \Pr[\mathbf{v} \text{ returned}] &= \prod_{i=1}^k \Pr[a_i = \mathbf{v}(A_i), \mathbf{E1}(i), \mathbf{E2}(i) \mid \mathbf{w}_{i-1} = \mathbf{v}[\mathcal{V}_{i-1}]] \\
 &\quad (\text{if } i = 1, \text{ then } \mathbf{w}_{i-1} = \mathbf{v}[\mathcal{V}_{i-1}] \text{ becomes } \mathbf{w}_0 = \mathbf{v}[\emptyset], \text{ which is vacuously true}) \\
 &= \prod_{i=1}^k \left( \Pr[a_i = \mathbf{v}(A_i), \mathbf{E1}(i) \mid \mathbf{w}_{i-1} = \mathbf{v}[\mathcal{V}_{i-1}]] \cdot \Pr[\mathbf{E2}(i) \mid \mathbf{E1}(i), a_i = \mathbf{v}(A_i), \mathbf{w}_{i-1} = \mathbf{v}[\mathcal{V}_{i-1}]] \right). \tag{15}
 \end{aligned}$$

Observe

$$\begin{aligned}
 &\Pr[a_i = \mathbf{v}(A_i), \mathbf{E1}(i) \mid \mathbf{w}_{i-1} = \mathbf{v}[\mathcal{V}_{i-1}]] \\
 &= \Pr[\mathbf{E1}(i) \mid \mathbf{w}_{i-1} = \mathbf{v}[\mathcal{V}_{i-1}]] \cdot \Pr[a_i = \mathbf{v}(A_i) \mid \mathbf{E1}(i), \mathbf{w}_{i-1} = \mathbf{v}[\mathcal{V}_{i-1}]] \\
 &= \frac{1}{|\text{DC}(A_i)|} \cdot \frac{|\sigma_{A_i=\mathbf{v}(A_i)}(\Pi_{\mathcal{Y}}(R_{F(\mathcal{X}^\circ, \mathcal{Y}^\circ)} \times \mathbf{v}[\mathcal{V}_{i-1}]))|}{|\Pi_{\mathcal{Y}}(R_{F(\mathcal{X}^\circ, \mathcal{Y}^\circ)} \times \mathbf{v}[\mathcal{V}_{i-1}]))|} \\
 &\quad (\text{note: } (\mathcal{X}^\circ, \mathcal{Y}^\circ, N_{\mathcal{Y}^\circ|\mathcal{X}^\circ}) = \text{constraint}_i^*(\mathbf{v}[\mathcal{V}_{i-1}], \mathbf{v}(A_i)), \text{ due to } \mathbf{E1}(i) \text{ and } \mathbf{w}_{i-1} = \mathbf{v}[\mathcal{V}_{i-1}]) \\
 &= \frac{1}{|\text{DC}(A_i)|} \cdot \text{reldeg}_{i,\mathcal{X}^\circ, \mathcal{Y}^\circ}(\mathbf{v}[\mathcal{V}_{i-1}], \mathbf{v}(A_i)) = \frac{1}{|\text{DC}(A_i)|} \cdot \text{reldeg}_i^*(\mathbf{v}[\mathcal{V}_{i-1}], \mathbf{v}(A_i)). \tag{16}
 \end{aligned}$$

On the other hand:

$$\begin{aligned}
& \Pr[\mathbf{E2}(i) \mid \mathbf{E1}(i), a_i = \mathbf{v}(A_i), \mathbf{w}_{i-1} = \mathbf{v}[\mathcal{V}_{i-1}]] \\
&= p_{\text{pass}}(i, \mathbf{v}[\mathcal{V}_{i-1}], \mathbf{v}[\mathcal{V}_i]) \\
\text{(by (12)) } &= \frac{B_i(\mathbf{v}[\mathcal{V}_i])}{B_{i-1}(\mathbf{v}[\mathcal{V}_{i-1}])} \cdot \frac{1}{\text{relddeg}_i^*(\mathbf{v}[\mathcal{V}_{i-1}], \mathbf{v}(A_i))}. \tag{17}
\end{aligned}$$

Plugging (16) and (17) into (15) yields

$$\begin{aligned}
\Pr[\mathbf{v} \text{ returned}] &= \prod_{i=1}^k \frac{B_i(\mathbf{v}[\mathcal{V}_i])}{B_{i-1}(\mathbf{v}[\mathcal{V}_{i-1}])} \cdot \frac{1}{|\text{DC}(A_i)|} = \frac{B_k(\mathbf{v}[\mathcal{V}_k])}{B_0(\mathbf{v}[\mathcal{V}_0])} \cdot \prod_{i=1}^k \frac{1}{|\text{DC}(A_i)|} \\
&= \frac{1}{B_0(\text{null})} \cdot \prod_{i=1}^k \frac{1}{|\text{DC}(A_i)|}.
\end{aligned}$$

As the above is identical for every  $\mathbf{v} \in \text{join}(\mathcal{Q})$ , we can conclude that each tuple in the join result gets returned by `ADC-sample` with the same probability. As an immediate corollary, each run of `ADC-sample` successfully returns a sample from  $\text{join}(\mathcal{Q})$  with probability

$$\frac{\text{OUT}}{B_0(\text{null})} \cdot \prod_{i=1}^k \frac{1}{|\text{DC}(A_i)|} \geq \frac{\text{OUT}}{\text{polymat}(\text{DC})} \cdot \prod_{i=1}^k \frac{1}{|\text{DC}(A_i)|} = \Omega\left(\frac{\text{OUT}}{\text{polymat}(\text{DC})}\right).$$

In the full version [37], we explain how to preprocess the relations of  $\mathcal{Q}$  in  $O(\text{IN})$  expected time to ensure that `ADC-sample` completes in  $O(1)$  time.

**Performing a Join Sampling Operation.** Recall that this operation must either return a uniformly random sample of  $\text{join}(\mathcal{Q})$  or declare  $\text{join}(\mathcal{Q}) = \emptyset$ . To support this operation, we execute two *threads* concurrently. The first thread repeatedly invokes `ADC-sample` until it successfully returns a sample. The other thread runs Ngo’s algorithm in [29] to compute  $\text{join}(\mathcal{Q})$  *in full*, after which we can declare  $\text{join}(\mathcal{Q}) \neq \emptyset$  or sample from  $\text{join}(\mathcal{Q})$  in constant time. As soon as one thread finishes, we manually terminate the other one.

This strategy guarantees that the join operation completes in  $O(\text{polymat}(\text{DC})/\max\{1, \text{OUT}\})$  time. To explain why, consider first the scenario where  $\text{OUT} \geq 1$ . In this case, we expect to find a sample with  $O(\text{polymat}(\text{DC})/\text{OUT})$  repeats of `ADC-sample`. Hence, the first thread finishes in  $O(\text{polymat}(\text{DC})/\text{OUT})$  expected sample time. On the other hand, if  $\text{OUT} = 0$ , the second thread will finish in  $O(\text{polymat}(\text{DC}))$  time. This concludes the proof of Theorem 3.

**Remarks.** When DC has only cardinality constraints (is thus “trivially” acyclic), `ADC-sample` simplifies into the sampling algorithm of Kim et al. [25]. In retrospect, two main obstacles prevent an obvious extension of their algorithm to an arbitrary acyclic DC. The first is identifying an appropriate way to deal with constraints  $(\mathcal{X}, \mathcal{Y}, N_{\mathcal{Y}|\mathcal{X}}) \in \text{DC}$  where  $\mathcal{X} \neq \emptyset$  (such constraints are absent in the degenerated context of [25]). The second obstacle involves determining how to benefit from a topological order (attribute ordering is irrelevant in [25]); replacing the order with a non-topological one may ruin the correctness of `ADC-sample`.

## 4 Directed Subgraph Sampling

Given a directed pattern graph  $P = (V_P, E_P)$  and a directed data graph  $G = (V, E)$ , we use  $\text{occ}(G, P)$  to represent the set of occurrences of  $P$  in  $G$ . Every vertex in  $G$  has an out-degree at most  $\lambda$ . Our goal is to design an algorithm to sample from  $\text{occ}(G, P)$  efficiently.

## 23:12 Join Sampling Under Acyclic Degree Constraints and (Cyclic) Subgraph Sampling

Let us formulate the “polymatroid bound” for this problem. Given an integer  $m \geq 1$ , an integer  $\lambda \in [1, m]$ , and a pattern  $P = (V_P, E_P)$ , first build a rule collection  $\mathcal{C}$  over  $V_P$  as follows: for each edge  $(X, Y) \in E_P$ , add to  $\mathcal{C}$  two rules:  $(\emptyset, \{X, Y\}, m)$  and  $(\{X\}, \{X, Y\}, \lambda)$ . Then, the *directed polymatroid bound* of  $m$ ,  $\lambda$ , and  $P$  can be defined as

$$\text{polymat}_{dir}(m, \lambda, P) = \text{polymat}(\mathcal{C}) \quad (18)$$

where  $\text{polymat}(\mathcal{C})$  follows the definition in (4).

This formulation reflects how directed subgraph listing can be processed as a join. Consider a *companion join*  $\mathcal{Q}$  constructed from  $G$  and  $P$  as follows. The schema graph of  $\mathcal{Q}$ , denoted as  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ , is exactly  $P = (V_P, E_P)$  (i.e.,  $\mathcal{V} = V_P$  and  $\mathcal{E} = E_P$ ). For every edge  $F = (X, Y) \in E_P$ , create a relation  $R_F \in \mathcal{Q}$  by inserting, for each edge  $(x, y)$  in the data graph  $G$ , a tuple  $\mathbf{u}$  with  $\mathbf{u}(X) = x$  and  $\mathbf{u}(Y) = y$  into  $R_F$ . The rule collection  $\mathcal{C}$  can now be regarded as a set DC of degree constraints with which  $\mathcal{Q}$  is consistent, i.e.,  $\mathcal{Q} \models \text{DC} = \mathcal{C}$ . The constraint dependence graph  $G_{\text{DC}}$  is precisely  $P$ . It is immediate that  $\text{polymat}_{dir}(|E|, \lambda, P) = \text{polymat}(\text{DC})$ . To find all the occurrences in  $\text{occ}(G, P)$ , it suffices to compute  $\text{join}(\mathcal{Q})$ . Specifically, every tuple  $\mathbf{u} \in \text{join}(\mathcal{Q})$  that uses a distinct value on every attribute in  $\mathcal{V}$  ( $= V_P$ ) matches a unique occurrence in  $\text{occ}(G, P)$ . Conversely, every occurrence in  $\text{occ}(G, P)$  matches the same number  $c$  of tuples in  $\text{join}(\mathcal{Q})$ , where  $c \geq 1$  is a constant equal to the number of automorphisms of  $P$ . If we denote  $\text{OUT} = |\text{occ}(G, P)|$  and  $\text{OUT}_{\mathcal{Q}} = |\text{join}(\mathcal{Q})|$ , it follows that  $c \cdot \text{OUT} \leq \text{OUT}_{\mathcal{Q}} \leq \text{polymat}(\text{DC}) = \text{polymat}_{dir}(|E|, \lambda, P)$ .

The above observation suggests how directed subgraph sampling can be reduced to join sampling. First, sample a tuple  $\mathbf{u}$  from  $\text{join}(\mathcal{Q})$  uniformly at random. Then, check whether  $\mathbf{u}(A) = \mathbf{u}(A')$  for any two distinct attributes  $A, A' \in \mathcal{V}$ . If so, declare failure; otherwise, declare success and return the unique occurrence matching the tuple  $\mathbf{u}$ . The success probability equals  $c \cdot \text{OUT} / \text{OUT}_{\mathcal{Q}}$ . In a success event, every occurrence in  $\text{occ}(G, P)$  has the same probability to be returned.

When  $P$  is acyclic, so is  $G_{\text{DC}}$ , and thus our algorithm in Theorem 3 can be readily applied to handle a subgraph sampling operation. To analyze the performance, consider first  $\text{OUT} \geq 1$ . We expect to draw  $O(\text{OUT}_{\mathcal{Q}} / \text{OUT})$  samples from  $\text{join}(\mathcal{Q})$  until a success event. As Theorem 3 guarantees retrieving a sample from  $\text{join}(\mathcal{Q})$  in  $O(\text{polymat}(\text{DC}) / \text{OUT}_{\mathcal{Q}})$  expected time, overall we expect to sample an occurrence from  $\text{occ}(G, P)$  in

$$O\left(\frac{\text{polymat}(\text{DC})}{\text{OUT}_{\mathcal{Q}}} \cdot \frac{\text{OUT}_{\mathcal{Q}}}{\text{OUT}}\right) = O\left(\frac{\text{polymat}(\text{DC})}{\text{OUT}}\right)$$

time. To prepare for the possibility of  $\text{OUT} = 0$ , we apply the “two-thread approach” in Section 3. We run a concurrent thread that executes Ngo’s algorithm in [29], which finds the whole  $\text{join}(\mathcal{Q})$ , and hence  $\text{occ}(G, P)$ , in  $O(\text{polymat}(\text{DC}))$  time, after which we can declare  $\text{occ}(G, P) = \emptyset$  or sample from  $\text{occ}(G, P)$  in  $O(1)$  time. By accepting whichever thread finishes earlier, we ensure that the operation completes in  $O(\text{polymat}(\text{DC}) / \max\{1, \text{OUT}\})$  time.

*The main challenge arises when  $P$  is cyclic.* In this case,  $G_{\text{DC}}$  (which equals  $P$ ) is cyclic. Thus, DC becomes a cyclic set of degree constraints, rendering neither Theorem 3 nor Ngo’s algorithm in [29] applicable. We overcome this challenge with the lemma below.

► **Lemma 5.** *If DC is cyclic, we can always find an acyclic subset  $\text{DC}' \subset \text{DC}$  satisfying  $\text{polymat}(\text{DC}') = \Theta(\text{polymat}(\text{DC}))$ .*

The proof is presented in Appendix A. Because  $\mathcal{Q} \models \text{DC}$  and  $\text{DC}'$  is a subset of DC, we know that  $\mathcal{Q}$  must be consistent with  $\text{DC}'$  as well, i.e.,  $\mathcal{Q} \models \text{DC}'$ . Therefore, our Theorem 3 can now be used to extract a sample from  $\text{join}(\mathcal{Q})$  in  $O(\text{polymat}_{dir}(\text{DC}') / \max\{1, \text{OUT}_{\mathcal{Q}}\})$

time. Importantly, Lemma 5 also permits us to directly apply Ngo’s algorithm in [29] to compute  $join(\mathcal{Q})$  in  $O(poly\text{mat}(\text{DC}'))$  time. Therefore, we can now apply the two-thread technique to sample from  $occ(G, P)$  in

$$O\left(\frac{poly\text{mat}(\text{DC}')}{\max\{1, \text{OUT}\}}\right) = O\left(\frac{poly\text{mat}(\text{DC})}{\max\{1, \text{OUT}\}}\right) = O\left(\frac{poly\text{mat}_{dir}(|E|, \lambda, P)}{\max\{1, \text{OUT}\}}\right)$$

time. We thus have arrived yet:

► **Theorem 6.** *Let  $G = (V, E)$  be a simple directed data graph, where each vertex has an out-degree at most  $\lambda$ . Let  $P = (V_P, E_P)$  be a simple weakly-connected directed pattern graph with a constant number of vertices. We can build in  $O(|E|)$  expected time a data structure that supports each subgraph sampling operation in  $O(poly\text{mat}_{dir}(|E|, \lambda, P) / \max\{1, \text{OUT}\})$  expected time, where  $\text{OUT}$  is the number of occurrences of  $P$  in  $G$ , and  $poly\text{mat}_{dir}(|E|, \lambda, P)$  is the directed polymatroid bound in (18).*

**Remarks.** For subgraph *listing*, Jayaraman et al. [18] presented a sophisticated method that also enables the application of Ngo’s algorithm in [29] to a cyclic  $P$ . Given the companion join  $\mathcal{Q}$ , they employ the *degree uniformization technique* [20] to generate  $t = O(\text{polylog } |E|)$  new joins  $\mathcal{Q}_1, \mathcal{Q}_2, \dots, \mathcal{Q}_t$  such that  $join(\mathcal{Q}) = \bigcup_{i=1}^t join(\mathcal{Q}_i)$ . For each  $i \in [t]$ , they construct an acyclic set  $\text{DC}_i$  of degree constraints (which is not always a subset of  $\text{DC}$ ) with the property  $\sum_{i=1}^t poly\text{mat}(\text{DC}_i) \leq poly\text{mat}(\text{DC})$ . Each join  $\mathcal{Q}_i$  ( $i \in [t]$ ) can then be processed by Ngo’s algorithm in  $O(poly\text{mat}(\text{DC}_i))$  time, thus giving an algorithm for computing  $join(\mathcal{Q})$  (and hence  $occ(G, P)$ ) in  $O(poly\text{mat}(\text{DC}))$  time. On the other hand, Lemma 5 facilitates a *direct* application of Ngo’s algorithm to  $\mathcal{Q}$ , implying the non-necessity of degree uniformization in subgraph listing. We believe that this simplification is noteworthy and merits its own dedicated exposition, considering the critical nature of the subgraph listing problem. In the absence of Lemma 5, integrating our join-sampling algorithm with the methodology of [18] for the purpose of subgraph sampling would require substantially more effort. Our proof of Lemma 5 *does* draw upon the analysis of [18], as discussed in depth in Appendix A.

## 5 Concluding Remarks

Our new sampling algorithms imply new results on several other fundamental problems. We will illustrate this with respect to evaluating a join  $\mathcal{Q}$  consistent with an acyclic set  $\text{DC}$  of degree constraints. Similar implications also apply to subgraph sampling.

- By standard techniques [10, 13], we can estimate the output size  $\text{OUT}$  up to a relative error  $\epsilon$  with high probability (i.e., at least  $1 - 1/\text{IN}^c$  for an arbitrarily large constant  $c$ ) in time  $\tilde{O}\left(\frac{1}{\epsilon^2} \frac{poly\text{mat}(\text{DC})}{\max\{1, \text{OUT}\}}\right)$  after a preprocessing of  $O(\text{IN})$  expected time.
- Employing a technique in [13], we can, with high probability, report all the tuples in  $join(\mathcal{Q})$  with a delay of  $\tilde{O}\left(\frac{poly\text{mat}(\text{DC})}{\max\{1, \text{OUT}\}}\right)$ . In this context, “delay” refers to the maximum interval between the reporting of two successive result tuples, assuming the presence of a placeholder tuple at the beginning and another at the end.
- In addition to the delay guarantee, our algorithm in the second bullet can, with high probability, report the tuples of  $join(\mathcal{Q})$  in a random permutation. This means that each of the  $\text{OUT}!$  possible permutations has an equal probability of being the output.

All of the results presented above compare favorably with the current state of the art as presented in [13]. This is primarily due to the superiority of  $poly\text{mat}(\text{DC})$  over  $AGM(\mathcal{Q})$ . In addition, our findings in the last two bullet points also complement Ngo’s algorithm as described in [29] in a satisfying manner.

## References

- 1 Amir Abboud, Seri Khoury, Oree Leibowitz, and Ron Safier. Listing 4-cycles. *CoRR*, abs/2211.10022, 2022. doi:10.48550/arXiv.2211.10022.
- 2 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
- 3 Swarup Acharya, Phillip B. Gibbons, Viswanath Poosala, and Sridhar Ramaswamy. Join synopses for approximate query answering. In *Proceedings of ACM Management of Data (SIGMOD)*, pages 275–286, 1999. doi:10.1145/304182.304207.
- 4 Noga Alon. On the number of subgraphs of prescribed type of graphs with a given number of edges. *Israel Journal of Mathematics*, 38:116–130, 1981.
- 5 Kaleb Alway, Eric Blais, and Semih Salihoglu. Box covers and domain orderings for beyond worst-case join processing. In *Proceedings of International Conference on Database Theory (ICDT)*, pages 3:1–3:23, 2021. doi:10.4230/LIPIcs.ICDT.2021.3.
- 6 Albert Aterias, Martin Grohe, and Daniel Marx. Size bounds and query plans for relational joins. *SIAM Journal on Computing*, 42(4):1737–1767, 2013. doi:10.1137/110859440.
- 7 Matthias Bentert, Till Fluschnik, Andre Nichterlein, and Rolf Niedermeier. Parameterized aspects of triangle enumeration. *Journal of Computer and System Sciences (JCSS)*, 103:61–77, 2019. doi:10.1016/j.jcss.2019.02.004.
- 8 Andreas Bjorklund, Rasmus Pagh, Virginia Vassilevska Williams, and Uri Zwick. Listing triangles. In *Proceedings of International Colloquium on Automata, Languages and Programming (ICALP)*, pages 223–234, 2014. doi:10.1007/978-3-662-43948-7\_19.
- 9 Surajit Chaudhuri, Rajeev Motwani, and Vivek R. Narasayya. On random sampling over joins. In *Proceedings of ACM Management of Data (SIGMOD)*, pages 263–274, 1999. doi:10.1145/304182.304206.
- 10 Yu Chen and Ke Yi. Random sampling and size estimation over cyclic joins. In *Proceedings of International Conference on Database Theory (ICDT)*, pages 7:1–7:18, 2020. doi:10.4230/LIPIcs.ICDT.2020.7.
- 11 N. Chiba and T. Nishizeki. Arboricity and subgraph listing algorithms. *SIAM Journal of Computing*, 14(1):210–223, 1985. doi:10.1137/0214017.
- 12 Kyle Deeds, Dan Suciu, Magda Balazinska, and Walter Cai. Degree sequence bound for join cardinality estimation. In *Proceedings of International Conference on Database Theory (ICDT)*, volume 255, pages 8:1–8:18, 2023. doi:10.4230/LIPIcs.ICDT.2023.8.
- 13 Shiyuan Deng, Shangqi Lu, and Yufei Tao. On join sampling and the hardness of combinatorial output-sensitive join algorithms. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 99–111, 2023. doi:10.1145/3584372.3588666.
- 14 David Eppstein. Subgraph isomorphism in planar graphs and related problems. *J. Graph Algorithms Appl.*, 3(3):1–27, 1999. doi:10.7155/jgaa.00014.
- 15 Hendrik Fichtenberger, Mingze Gao, and Pan Peng. Sampling arbitrary subgraphs exactly uniformly in sublinear time. In *Proceedings of International Colloquium on Automata, Languages and Programming (ICALP)*, pages 45:1–45:13, 2020. doi:10.4230/LIPIcs.ICALP.2020.45.
- 16 Tomasz Gogacz and Szymon Torunczyk. Entropy bounds for conjunctive queries with functional dependencies. In *Proceedings of International Conference on Database Theory (ICDT)*, volume 68, pages 15:1–15:17, 2017. doi:10.4230/LIPIcs.ICDT.2017.15.
- 17 Chinh T. Hoang, Marcin Kaminski, Joe Sawada, and R. Sritharan. Finding and listing induced paths and cycles. *Discrete Applied Mathematics*, 161(4-5):633–641, 2013. doi:10.1016/j.dam.2012.01.024.
- 18 Sai Vikneshwar Mani Jayaraman, Corey Ropell, and Atri Rudra. Worst-case optimal binary join algorithms under general  $\ell_p$  constraints. *CoRR*, abs/2112.01003, 2021. doi:10.48550/arXiv.2112.01003.
- 19 Ce Jin and Yinzhao Xu. Removing additive structure in 3sum-based reductions. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, pages 405–418, 2023. doi:10.1145/3564246.3585157.

- 20 Manas Joglekar and Christopher Re. It's all a matter of degree - using degree information to optimize multiway joins. *Theory Comput. Syst.*, 62(4):810–853, 2018. doi:10.1007/s00224-017-9811-8.
- 21 Mahmoud Abo Khamis, Vasileios Nakos, Dan Olteanu, and Dan Suciu. Join size bounds using lp-norms on degree sequences. *CoRR*, abs/2306.14075, 2023. doi:10.48550/arXiv.2306.14075.
- 22 Mahmoud Abo Khamis, Hung Q. Ngo, Christopher Re, and Atri Rudra. Joins via geometric resolutions: Worst case and beyond. *ACM Transactions on Database Systems (TODS)*, 41(4):22:1–22:45, 2016. doi:10.1145/2967101.
- 23 Mahmoud Abo Khamis, Hung Q. Ngo, and Dan Suciu. Computing join queries with functional dependencies. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 327–342, 2016. doi:10.1145/2902251.2902289.
- 24 Mahmoud Abo Khamis, Hung Q. Ngo, and Dan Suciu. What do shannon-type inequalities, submodular width, and disjunctive datalog have to do with one another? In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 429–444, 2017. doi:10.1145/3034786.3056105.
- 25 Kyoungmin Kim, Jaehyun Ha, George Fletcher, and Wook-Shin Han. Guaranteeing the  $\tilde{O}(\text{AGM}/\text{OUT})$  runtime for uniform sampling and size estimation over joins. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 113–125, 2023. doi:10.1145/3584372.3588676.
- 26 George Manoussakis. Listing all fixed-length simple cycles in sparse graphs in optimal time. In *Fundamentals of Computation Theory*, pages 355–366, 2017. doi:10.1007/978-3-662-55751-8\_28.
- 27 Gonzalo Navarro, Juan L. Reutter, and Javiel Rojas-Ledesma. Optimal joins using compact data structures. In *Proceedings of International Conference on Database Theory (ICDT)*, volume 155, pages 21:1–21:21, 2020. doi:10.4230/LIPIcs.ICDT.2020.21.
- 28 Jaroslav Nesetril and Svatopluk Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 26(2):415–419, 1985.
- 29 Hung Q. Ngo. Worst-case optimal join algorithms: Techniques, results, and open problems. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 111–124, 2018. doi:10.1145/3196959.3196990.
- 30 Hung Q. Ngo, Dung T. Nguyen, Christopher Re, and Atri Rudra. Beyond worst-case analysis for joins with minesweeper. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 234–245, 2014. doi:10.1145/2594538.2594547.
- 31 Hung Q. Ngo, Ely Porat, Christopher Ré, and Atri Rudra. Worst-Case Optimal Join Algorithms: [Extended Abstract]. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 37–48, 2012. doi:10.1145/2213556.2213565.
- 32 Hung Q. Ngo, Ely Porat, Christopher Re, and Atri Rudra. Worst-case optimal join algorithms. *Journal of the ACM (JACM)*, 65(3):16:1–16:40, 2018. doi:10.1145/3180143.
- 33 Hung Q. Ngo, Christopher Re, and Atri Rudra. Skew strikes back: new developments in the theory of join algorithms. *SIGMOD Rec.*, 42(4):5–16, 2013. doi:10.1145/2590989.2590991.
- 34 Dan Suciu. Applications of information inequalities to database theory problems. *CoRR*, abs/2304.11996, 2023. doi:10.48550/arXiv.2304.11996.
- 35 Maciej M. Syslo. An efficient cycle vector space algorithm for listing all cycles of a planar graph. *SIAM Journal of Computing*, 10(4):797–808, 1981. doi:10.1137/0210062.
- 36 Todd L. Veldhuizen. Triejoin: A simple, worst-case optimal join algorithm. In *Proceedings of International Conference on Database Theory (ICDT)*, pages 96–106, 2014. doi:10.5441/002/icdt.2014.13.
- 37 Ru Wang and Yufei Tao. Join sampling under acyclic degree constraints and (cyclic) subgraph sampling, 2023. doi:10.48550/arXiv.2312.12797.
- 38 Zhuoyue Zhao, Robert Christensen, Feifei Li, Xiao Hu, and Ke Yi. Random sampling over joins revisited. In *Proceedings of ACM Management of Data (SIGMOD)*, pages 1525–1539, 2018. doi:10.1145/3183713.3183739.



## A Proof of Lemma 5

Let us rephrase the problem as follows. Let  $P = (V_P, E_P)$  be a *cyclic* pattern graph. Given an integer  $m \geq 1$  and an integer  $\lambda \in [1, m]$ , define DC to be a set of degree constraints over  $V_P$  that contains two constraints for each edge  $(X, Y) \in E_P$ :  $(\emptyset, \{X, Y\}, m)$  and  $(\{X\}, \{X, Y\}, \lambda)$ . The constraint dependence graph  $G_{\text{DC}}$  is exactly  $P$  and, hence, is cyclic. We want to prove the existence of an acyclic  $\text{DC}' \subset \text{DC}$  such that  $\text{polymat}(\text{DC}') = \text{polymat}(\text{DC})$ . We will first tackle the situation where  $\lambda > \sqrt{m}$  before proceeding to the opposite scenario. The former case presents a more intriguing line of argumentation than the latter.

### A.1 Case $\lambda > \sqrt{m}$

For every edge  $(X, Y) \in G_{\text{DC}} = (V_P, E_P)$ , define two variables:  $x_{X,Y}$  and  $z_{X,Y}$ . Jayaraman et al. [18] showed that, for  $\lambda > \sqrt{m}$ ,  $\text{polymat}(\text{DC})$  is, up to a constant factor, the optimal value of the following LP (named  $\text{LP}^{(+)}$  following [18]):

$$\begin{aligned} \text{LP}^{(+)} [18] \quad \min \quad & \sum_{(X,Y) \in E_P} x_{X,Y} \log m + z_{X,Y} \log \lambda \quad \text{subject to} \\ & \sum_{(X,A) \in E_P} (x_{X,A} + z_{X,A}) + \sum_{(A,Y) \in E_P} x_{A,Y} \geq 1 \quad \forall A \in V_P \\ & x_{X,Y} \geq 0, z_{X,Y} \geq 0 \quad \forall (X,Y) \in E_P \end{aligned}$$

► **Lemma 7.** *There exists an optimal solution to  $\text{LP}^{(+)}$  satisfying the condition that the edges in  $\{(X, Y) \in E_P \mid z_{X,Y} > 0\}$  induce an acyclic subgraph of  $G_{\text{DC}}$ .*

We note that while the above lemma is not expressly stated in [18], it can be extrapolated from the analysis presented in Section H.2 of [18]. Nevertheless, the argument laid out in [18] is quite intricate. Our proof, which will be presented below, incorporates new ideas beyond their argument and is considerably shorter. Specifically, these new ideas are evidenced in the way we formulate a novel LP optimal solution in (19)-(22).

**Proof of Lemma 7.** Consider an arbitrary optimal solution to  $\text{LP}^{(+)}$  that sets  $x_{X,Y} = x_{X,Y}^*$  and  $z_{X,Y} = z_{X,Y}^*$  for each  $(X, Y) \in E_P$ . If the edge set  $\{(X, Y) \in E_P \mid z_{X,Y}^* > 0\}$  induces an acyclic graph, we are done. Next, we consider that  $G_{\text{DC}}$  contains a cycle.

Suppose that  $(A_1, A_2)$  is the edge in the cycle with the smallest  $z_{A_1, A_2}^*$  (breaking ties arbitrarily). Let  $(A_2, A_3)$  be the edge succeeding  $(A_1, A_2)$  in the cycle. It thus follows that  $z_{A_2, A_3}^* \geq z_{A_1, A_2}^*$ . Define

$$x'_{A_2, A_3} = x_{A_2, A_3}^* + z_{A_1, A_2}^* \tag{19}$$

$$x'_{A_1, A_2} = x_{A_1, A_2}^* \tag{20}$$

$$z'_{A_2, A_3} = 0 \tag{21}$$

$$z'_{A_1, A_2} = 0 \tag{22}$$

For every edge  $(X, Y) \in E_P \setminus \{(A_1, A_2), (A_2, A_3)\}$ , set  $x'_{X,Y} = x_{X,Y}^*$  and  $z'_{X,Y} = z_{X,Y}^*$ . It is easy to verify that, for every vertex  $A \in V_P$ , we have

$$\sum_{(X,A) \in E_P} (x'_{X,A} + z'_{X,A}) + \sum_{(A,Y) \in E_P} x'_{A,Y} \geq \sum_{(X,A) \in E_P} (x_{X,A}^* + z_{X,A}^*) + \sum_{(A,Y) \in E_P} x_{A,Y}^*.$$

Therefore,  $\{x'_{X,Y}, z'_{X,Y} \mid (X, Y) \in E_P\}$  serves as a feasible solution to  $\text{LP}^{(+)}$ . However:



$$\begin{aligned}
& \left( \sum_{(X,Y) \in E_P} x'_{X,Y} \log m + z'_{X,Y} \log \lambda \right) - \left( \sum_{(X,Y) \in E_P} x^*_{X,Y} \log m + z^*_{X,Y} \log \lambda \right) \\
&= z^*_{A_1, A_2} \log m - (z^*_{A_1, A_2} + z^*_{A_2, A_3}) \log \lambda \\
&\leq z^*_{A_1, A_2} \log m - 2 \cdot z^*_{A_1, A_2} \log \lambda \\
&< 0
\end{aligned} \tag{23}$$

where the last step used the fact  $\lambda^2 > m$ . This contradicts the optimality of  $\{x^*_{X,Y}, z^*_{X,Y} \mid (X,Y) \in E_P\}$ .  $\blacktriangleleft$

We now build a set  $\text{DC}'$  of degree constraints as follows. First, take an optimal solution  $\{x^*_{X,Y}, z^*_{X,Y} \mid (X,Y) \in E_P\}$  to  $\text{LP}^{(+)}$  promised by Lemma 7. Add to  $\text{DC}'$  a constraint  $(X, \{X, Y\}, \lambda)$  for every  $(X, Y) \in E_P$  satisfying  $z^*_{X,Y} > 0$ . Then, for every edge  $(X, Y) \in E_P$ , add to  $\text{DC}'$  a constraint  $(\emptyset, \{X, Y\}, m)$ . The  $\text{DC}'$  thus constructed must be acyclic. Denote by  $G_{\text{DC}'} = (V'_P, E'_P)$  the degree constraint graph of  $\text{DC}'$ . Note that  $V_P = V'_P$  and  $E'_P \subset E_P$ .

**► Lemma 8.** *The  $\text{DC}'$  constructed in the above manner satisfies  $\text{polymat}(\text{DC}') = \Theta(\text{polymat}(\text{DC}))$ .*

**Proof.** We will first establish  $\text{polymat}(\text{DC}') \geq \text{polymat}(\text{DC})$ . Remember that  $\text{polymat}(\text{DC}')$  is the optimal value of the modular LP (in its primal form) defined by  $\text{DC}'$ , as described in Section 2. Similarly,  $\text{polymat}(\text{DC})$  is the optimal value of the modular LP defined by  $\text{DC}$ . Given that  $\text{DC}' \subset \text{DC}$ , the LP defined by  $\text{DC}'$  incorporates only a subset of the constraints found in the LP defined by  $\text{DC}$ . Therefore, it must be the case that  $\text{polymat}(\text{DC}') \geq \text{polymat}(\text{DC})$ .

The rest of the proof will show  $\text{polymat}(\text{DC}') = O(\text{polymat}(\text{DC}))$ , which will establish the lemma. Consider the following LP:

$$\begin{aligned}
\text{LP}_1^{(+)} \quad & \min \sum_{(X,Y) \in E_P} x_{X,Y} \log m + z_{X,Y} \log \lambda \text{ subject to} \\
& \sum_{(X,A) \in E_P} x_{X,A} + \sum_{(A,Y) \in E_P} x_{A,Y} + \sum_{(X,A) \in E'_P} z_{X,A} \geq 1 \quad \forall A \in \mathcal{V}_P \\
& x_{X,Y} \geq 0, z_{X,Y} \geq 0 \quad \forall (X,Y) \in E_P
\end{aligned}$$

The condition  $(X, A) \in E'_P$  in the first inequality marks the difference between  $\text{LP}_1^{(+)}$  and  $\text{LP}^{(+)}$ . Note that the two LPs have the same objective function.

**► Claim 1.**  $\text{LP}_1^{(+)}$  and  $\text{LP}^{(+)}$  have the same optimal value.

To prove the claim, first observe that any feasible solution  $\{x_{X,Y}, z_{X,Y} \mid (X,Y) \in E_P\}$  to  $\text{LP}_1^{(+)}$  is also a feasible solution to  $\text{LP}^{(+)}$ . Hence, the optimal value of  $\text{LP}^{(+)}$  cannot exceed that of  $\text{LP}_1^{(+)}$ . On the other hand, recall that earlier we have identified an optimal solution  $\{x^*_{X,Y}, z^*_{X,Y} \mid (X,Y) \in E_P\}$  to  $\text{LP}^{(+)}$ . By how  $\text{DC}'$  is built from that solution and how  $G_{\text{DC}'} = (V'_P, E'_P)$  is built from  $\text{DC}'$ , it must hold that  $z^*_{X,Y} = 0$  for every  $(X, Y) \in E_P \setminus E'_P$ . Hence,  $\{x^*_{X,Y}, z^*_{X,Y} \mid (X, Y) \in E_P\}$  makes a feasible solution to  $\text{LP}_1^{(+)}$ . This implies that  $\{x^*_{X,Y}, z^*_{X,Y} \mid (X, Y) \in E_P\}$  must be an optimal solution to  $\text{LP}_1^{(+)}$ . Claim 1 now follows.

Consider another LP:

$$\begin{aligned}
 \mathbf{LP}_2^{(+)} \quad & \min \sum_{(X,Y) \in E_P} x_{X,Y} \log m + \sum_{(X,Y) \in E'_P} z_{X,Y} \log \lambda \text{ subject to} \\
 & \sum_{(X,A) \in E_P} x_{X,A} + \sum_{(A,Y) \in E_P} x_{A,Y} + \sum_{(X,A) \in E'_P} z_{X,A} \geq 1 \quad \forall A \in \mathcal{V}_P \\
 & x_{X,Y} \geq 0 \quad \forall (X,Y) \in E_P \\
 & z_{X,Y} \geq 0 \quad \forall (X,Y) \in E'_P
 \end{aligned}$$

$\mathbf{LP}_2^{(+)}$  differs from  $\mathbf{LP}_1^{(+)}$  in that the former drops the variables  $z_{X,Y}$  of those edges  $(X,Y) \in E_P \setminus E'_P$ . This happens both in the constraints and the objective function.

▷ **Claim 2.**  $\mathbf{LP}_1^{(+)}$  and  $\mathbf{LP}_2^{(+)}$  have the same optimal value.

To prove the claim, first observe that, given a feasible solution  $\{x_{X,Y} \mid (X,Y) \in E_P\} \cup \{z_{X,Y} \mid (X,Y) \in E'_P\}$  to  $\mathbf{LP}_2^{(+)}$ , we can extend it into a feasible solution to  $\mathbf{LP}_1^{(+)}$  by padding  $Z_{X,Y} = 0$  for each  $(X,Y) \in E_P \setminus E'_P$ . Hence, the optimal value of  $\mathbf{LP}_1^{(+)}$  cannot exceed that of  $\mathbf{LP}_2^{(+)}$ . On the other hand, as mentioned before,  $\{x_{X,Y}^*, z_{X,Y}^* \mid (X,Y) \in E_P\}$  is an optimal solution to  $\mathbf{LP}_1^{(+)}$ . In this solution,  $z_{X,Y}^* = 0$  for every  $(X,Y) \in E_P \setminus E'_P$ . Thus,  $\{x_{X,Y}^* \mid (X,Y) \in E_P\} \cup \{z_{X,Y}^* \mid (X,Y) \in E'_P\}$  makes a feasible solution to  $\mathbf{LP}_2^{(+)}$ , achieving the same objective function value as the optimal value of  $\mathbf{LP}_1^{(+)}$ . Claim 2 now follows.

Finally, notice that  $\mathbf{LP}_2^{(+)}$  is exactly the dual modular LP defined by  $\mathbf{DC}'$ . Hence,  $\log(\text{polymat}(\mathbf{DC}'))$  is exactly the optimal value of  $\mathbf{LP}_2^{(+)}$ . Thus,  $\text{polymat}(\mathbf{DC}') = O(\text{polymat}(\mathbf{DC}))$  can now be derived from the above discussion and the fact that  $\log(\text{polymat}(\mathbf{DC}))$  is asymptotically the optimal value of  $\mathbf{LP}^{(+)}$ . ◀

## A.2 Case $\lambda \leq \sqrt{m}$

Let us first define several concepts. A *directed star* refers to a directed graph where there are  $t \geq 2$  vertices, among which one vertex, designated the *center*, has  $t - 1$  edges (in-coming and out-going edges combined), and every other vertex, called a *petal*, has only one edge (which can be an in-coming or out-going edge). Now, consider a directed bipartite graph between  $U_1$  and  $U_2$ , each being an independent sets of vertices (an edge may point from one vertex in  $U_1$  to a vertex in  $U_2$ , or vice versa). A *directed star cover* of the bipartite graph is a set of directed stars such that

- each directed star is a subgraph of the bipartite graph,
- no two directed stars share a common edge, and
- every vertex in  $U_1 \cup U_2$  appears in exactly one directed star.

A directed star cover is *minimum* if it has the least number of edges, counting all directed stars in the cover.

Next, we review an expression about  $\text{polymat}(\mathbf{DC})$  derived in [18]. Find all the strongly connected components (SCCs) of  $G_{\mathbf{DC}} = (V_P, E_P)$ . Adopting terms from [18], an SCC is classified as (i) a *source* if it has no in-coming edge from another SCC, or a *non-source* otherwise; (ii) *trivial* if it consists of a single vertex, or *non-trivial* otherwise. Define:

- $S$  = the set of vertices in  $G_{\mathbf{DC}}$  each forming a trivial source SCC by itself.
  - $T$  = the set of vertices in  $G_{\mathbf{DC}}$  receiving an in-coming edge from at least one vertex in  $S$ .
- Take a minimum directed star cover of the directed bipartite graph induced by  $S$  and  $T$ . Define
- $S_1$  = the set of vertices in  $S$  each serving as the center of some directed star in the cover.
  - $S_2 = S \setminus S_1$ .

- $T_2$  = the set of vertices in  $T$  each serving as the center of some directed star in the cover.
- $T_1 = T \setminus T_2$ .

Note that the meanings of the symbols  $S_1, S_2, T_1$ , and  $T_2$  follow exactly those in [18] for the reader's convenience (in particular, note the semantics of  $T_1$  and  $T_2$ ).

We now introduce three quantities:

- $c_1$ : the number of non-trivial source SCCs;
- $n_1$ : the total number of vertices in non-trivial source SCCs;
- $n_2 = |V_P| - n_1 - |S| - |T|$ .

Jayaraman et al. [18] showed:

$$\text{polymat}_{\text{dir}}(m, \lambda, P) = \Theta\left(m^{c_1+|S|} \cdot \lambda^{n_1+n_2+|T_1|-2c_1-|S_1|}\right). \quad (24)$$

Let  $G'_{\text{DC}} = (V'_P, E'_P)$  be an arbitrary weakly-connected acyclic subgraph of  $G_{\text{DC}}$  satisfying all the conditions below.

- $V_P = V'_P$ .
- $E'_P$  contains all the edges in the minimum directed star cover identified earlier.
- In each non-trivial source SCC, every vertex, except for one, has one in-coming edge included in  $E'_P$ . We will refer to the vertex  $X$  with no in-coming edges in  $E'_P$  as the SCC's *root*. The fact that every other vertex  $Y$  in the SCC has an in-coming edge in  $E'_P$  implies  $(X, Y) \in E'_P$  for at least one  $Y$ . We designate one such  $(X, Y)$  as the SCC's *main edge*.
- In each non-trivial non-source SCC, every vertex has an in-coming edge included in  $E'_P$ .

It is rudimentary to verify that such a subgraph  $G'_{\text{DC}}$  must exist.

From  $G_{\text{DC}} = (V_P, E_P)$  and  $G'_{\text{DC}} = (V'_P, E'_P)$ , we create a set  $\text{DC}'$  of degree constraints as follows.

- For each edge  $(X, Y) \in E_P$  (note: not  $E'_P$ ), add a constraint  $(\emptyset, \{X, Y\}, m)$  to  $\text{DC}'$ .
- We inspect each directed star in the minimum directed star cover and distinguish two possibilities.
  - Scenario 1: The star's center  $X$  comes from  $S_1$ . Let the star's petals be  $Y_1, Y_2, \dots, Y_t$  for some  $t \geq 1$ ; the ordering of the petals does not matter. For each  $i \in [t-1]$ , we add a constraint  $(\{X\}, \{X, Y_i\}, \lambda)$  to  $\text{DC}'$ . We will refer to  $(X, Y_i)$  as the star's *main edge*.
  - Scenario 2: The star's center  $X$  comes from  $T_2$ . Nothing needs to be done.
- Consider now each non-trivial source SCC. Remember that every vertex  $Y$ , other than the SCC's root, has an in-coming edge  $(X, Y) \in E'_P$ . For every such  $Y$ , if  $(X, Y)$  is not the SCC's main edge, add a constraint  $(\{X\}, \{X, Y\}, \lambda)$  to  $\text{DC}'$ .
- Finally, we examine each non-source SCC. As mentioned, every vertex  $Y$  in such an SCC has an in-coming edge  $(X, Y) \in E'_P$ . For every  $Y$ , add a constraint  $(\{X\}, \{X, Y\}, \lambda)$  to  $\text{DC}'$ .

The rest of the proof will show  $\text{polymat}(\text{DC}') = \Theta(\text{polymat}(\text{DC}))$ . As  $\text{DC}' \subset \text{DC}$ , we must have  $\text{polymat}(\text{DC}') \geq \text{polymat}(\text{DC})$  following the same reasoning used in the  $\lambda > \sqrt{m}$  case.

We will now proceed to argue that  $\text{polymat}(\text{DC}') = O(\text{polymat}(\text{DC}))$ . Recall that  $\log(\text{polymat}(\text{DC}'))$  is the optimal value of the dual modular LP of  $\text{DC}'$  (see Section 2). On the other hand, the value of  $\text{polymat}(\text{DC})$  satisfies (24). In the following, we will construct a feasible solution to the dual modular LP of  $\text{DC}'$  under which the LP's objective function achieves the value of

$$\left((c_1 + |S|) \cdot \log m\right) + (n_1 + n_2 + |T_1| - 2c_1 - |S_1|) \cdot \log \lambda \quad (25)$$

which will be sufficient for proving Lemma 7.

## 23:20 Join Sampling Under Acyclic Degree Constraints and (Cyclic) Subgraph Sampling

The dual modular LP associates every constraint  $(\mathcal{X}, \mathcal{Y}, N_{\mathcal{Y}|\mathcal{X}}) \in \text{DC}'$  with a variable  $\delta_{\mathcal{Y}|\mathcal{X}}$ . We determine these variables' values as follows.

- For every constraint  $(\mathcal{X}, \mathcal{Y}, N_{\mathcal{Y}|\mathcal{X}}) \in \text{DC}'$  where  $N_{\mathcal{Y}|\mathcal{X}} = \lambda$ , set  $\delta_{\mathcal{Y}|\mathcal{X}} = 1$ .
- Consider each directed star in the minimum directed star.
  - Scenario 1: The star's center  $X$  comes from  $S_1$ . For the star's main edge  $(X, Y)$ , the constraint  $(\emptyset, \{X, Y\}, m)$  exists in  $\text{DC}'$ . Set  $\delta_{\{X, Y\}|\emptyset} = 1$ .
  - Scenario 2: The star's center  $X$  comes from  $T_2$ . For every petal  $Y$  of the star, the constraint  $(\emptyset, \{X, Y\}, m)$  exists in  $\text{DC}'$ . Set  $\delta_{\{X, Y\}|\emptyset} = 1$ .
- Consider each non-trivial source SCC. Let  $(X, Y)$  be the main edge of the SCC. The constraint  $(\emptyset, \{X, Y\}, m)$  exists in  $\text{DC}'$ . Set  $\delta_{\{X, Y\}|\emptyset} = 1$ .

The other variables that have not yet been mentioned are all set to 0.

It is tedious but straightforward to verify that all the constraints of the dual modular LP are fulfilled. To confirm that the objective function indeed evaluates to (25), observe:

- There are  $c_1 + |S|$  constraints of the form  $(\emptyset, \{X, Y\}, m)$  with  $\delta_{\{X, Y\}|\emptyset} = 1$ . Specifically,  $c_1$  of them come from the roots of the non-trivial source SCCs,  $|S_1|$  of them come from the star center vertices in  $S_1$ , and  $|S_2|$  of them come from the petal vertices in  $S_2$ .
- There are  $n_1 + n_2 + |T_1| - 2c_1 - |S_1|$  of the form  $(\{X\}, \{X, Y\}, \lambda)$  with  $\delta_{\{X, Y\}|\{X\}} = 1$ . Specifically,  $n_1 - 2c_1$  of them come from the non-main edges of the non-trivial source SCCs,  $n_2$  of them come from the vertices that are not in any non-trivial source SCC and are not in  $S \cup T$ , and  $|T_1| - |S_1|$  of them come from the petal vertices that are (i) in  $T_1$  but (ii) not in the main edges of their respective stars.

We now conclude the whole proof of Lemma 5.

# Finding Smallest Witnesses for Conjunctive Queries

Xiao Hu 

University of Waterloo, Canada

Stavros Sintos 

University of Illinois Chicago, IL, USA

---

## Abstract

A witness is a sub-database that preserves the query results of the original database but of much smaller size. It has wide applications in query rewriting and debugging, query explanation, IoT analytics, multi-layer network routing, etc. In this paper, we study the smallest witness problem (SWP) for the class of conjunctive queries (CQs) without self-joins.

We first establish the dichotomy that SWP for a CQ can be computed in polynomial time if and only if it has *head-cluster property*, unless  $P = NP$ . We next turn to the approximated version by relaxing the size of a witness from being minimum. We surprisingly find that the *head-domination property* – that has been identified for the deletion propagation problem [31] – can also precisely capture the hardness of the approximated smallest witness problem. In polynomial time, SWP for any CQ with head-domination property can be approximated within a constant factor, while SWP for any CQ without such a property cannot be approximated within a logarithmic factor, unless  $P = NP$ .

We further explore efficient approximation algorithms for CQs without head-domination property: (1) we show a trivial algorithm which achieves a polynomially large approximation ratio for general CQs; (2) for any CQ with only one non-output attribute, such as star CQs, we show a greedy algorithm with a logarithmic approximation ratio; (3) for line CQs, which contain at least two non-output attributes, we relate SWP problem to the directed steiner forest problem, whose algorithms can be applied to line CQs directly. Meanwhile, we establish a much higher lower bound, exponentially larger than the logarithmic lower bound obtained above. It remains open to close the gap between the lower and upper bound of the approximated SWP for CQs without head-domination property.

**2012 ACM Subject Classification** Theory of computation → Data provenance

**Keywords and phrases** conjunctive query, smallest witness, head-cluster, head-domination

**Digital Object Identifier** 10.4230/LIPIcs.ICDT.2024.24

**Related Version** *Full Version*: <https://arxiv.org/abs/2311.18157>

## 1 Introduction

To deal with large-scale data in analytical applications, people have developed a large body of data summarization techniques to reduce computational as well as storage complexity, such as sampling [10, 41, 12], sketch [15], coresets [36] and factorization [34]. The notion of witness has been studied as one form of why-provenance [9, 26, 3] that provides a *proof* for output results, with wide applications in explainable data-intensive analytics. The *smallest witness problem* was first proposed by [32] that given a query  $Q$ , a database  $D$  and one specific query result  $t \in Q(D)$ , the target is to find the smallest sub-database  $D' \subseteq D$  such that  $t$  is witnessed by  $D'$ , i.e.,  $t \in Q(D')$ . In this paper, we consider a generalized notion for all query results, i.e., our target is to find the smallest sub-database  $D' \subseteq D$  such that all query results can be witnessed by  $D'$ , i.e.,  $Q(D) = Q(D')$ . Our generalized smallest witness has many useful applications in practice, such as helping students learn SQL queries [32], query rewriting, query explanation, multi-layer network routing, IoT analytics on edge devices [35]. We mention three application scenarios:



© Xiao Hu and Stavros Sintos;

licensed under Creative Commons License CC-BY 4.0

27th International Conference on Database Theory (ICDT 2024).

Editors: Graham Cormode and Michael Shekelyan; Article No. 24; pp. 24:1–24:20

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

► **Example 1.** Alice located at Seattle wants to send the query results of  $Q$  over a database  $D$  (which is also stored at Seattle) to Bob located at New York. Unfortunately, the number of query results could be polynomially large in terms of the number of tuples in  $D$ . An alternative is to send the entire database  $D$ , since Bob can retrieve all query results by executing  $Q$  over  $D$  at New York. However, moving the entire database is also expensive. A natural question arises: is it necessary for Alice to send the entire  $D$  or  $Q(D)$ ? If not, what is the smallest subset of tuples to send?

► **Example 2.** Charlie is a novice at learning SQL in a undergraduate database course. Suppose there is a huge test database  $D$ , a correct query  $Q$  that Charlie is expected to learn, and a wrong query  $Q'$  submitted by him, where some answers in  $Q(D)$  are missed from  $Q'(D)$ . To help Charlie debug, the instructor can simply show the whole test database  $D$  to him. However, Charlie will have to dive into such a huge database to figure out where his query goes wrong. A natural question arises: is it necessary to show the entire  $D$  to Charlie? If not, what is the smallest subset of tuples to show so that Charlie can quickly find all missing answers by his wrong query?

► **Example 3.** In a multi-layer communication network, clients and servers are connected by routers organized into layers, such that links (or edges) exist between routers residing in consecutive layers. What is the smallest subset of links needed for building a fully connected network, i.e., every client-server pair is connected via a directed path? For a given network, what is the maximum number of links that can be broken while the connectivity with respect to the client-server pairs does not change? This information could help evaluate the inherent robustness of a network to either malicious attacks or even just random failures.

Recall that our smallest witness problem finds the smallest sub-database  $D' \subseteq D$  such that  $Q(D) = Q(D')$ . It would be sufficient for Alice to send  $D'$ , while Bob can retrieve all query results by executing  $Q$  over  $D'$ , saving much transmission cost. Also, it would be sufficient for the instructor to show  $D'$  to Charlie, from which all correct answers in  $Q(D)$  can be recovered, saving Charlie much efforts in exploring a huge test database.<sup>1</sup> Moreover, the connectivity of a multi-layer network  $D$  can be modeled as a line query  $Q$  (formally defined in Section 5.3) with connected client-server pairs as  $Q(D)$ , such that  $D'$  is a smallest subset of links needed for maintaining the desired connectivity, and  $D - D'$  is a maximum subset of links that can be removed safely. In this paper, we aim to design algorithms that can efficiently compute or approximate the smallest witness for conjunctive queries, and understand the hardness of this problem when such algorithms do not exist.

## 1.1 Problem Definition

Let  $\mathbb{R}$  be a database schema that contains  $m$  relations  $R_1, R_2, \dots, R_m$ . Let  $\mathbb{A}$  be the set of all attributes in  $\mathbb{R}$ . Each relation  $R_i$  is defined on a subset of attributes  $\mathbb{A}_i \subseteq \mathbb{A}$ . We use  $A, B, C, A_1, A_2, A_3, \dots$  etc. to denote the attributes in  $\mathbb{A}$  and  $a, b, c, \dots$  etc. to denote their values. Let  $\text{dom}(A)$  be the domain of attribute  $A \in \mathbb{A}$ . The domain of a set of attributes  $X \subseteq \mathbb{A}$  is defined as  $\text{dom}(X) = \prod_{A \in X} \text{dom}(A)$ . Given the database schema  $\mathbb{R}$ , let  $D$  be a given database of  $\mathbb{R}$ , and let the corresponding relations of  $R_1, \dots, R_m$  be  $R_1^D, \dots, R_m^D$ , where  $R_i^D$  is a collection of tuples defined on  $\text{dom}(\mathbb{A}_i)$ . The *input size* of database  $D$  is denoted as  $N = |D| = \sum_{i \in [m]} |R_i^D|$ . Where  $D$  is clear from the context, we will drop the superscript.

<sup>1</sup> The smallest witness for a single query result [32] has been incorporated into an educational tool (<https://dukedb-hnrq.github.io/>), successfully employed in Duke database courses with 1,000+ undergraduate users. Our generalized version can also be incorporated and save more efforts by showing one small witness for all answers.

$R_1$		$R_2$		$R_3$		$R_4$		$Q(D)$		
A	B	B	C	C	F	C	H	A	C	F
a1	b1	b1	c1	c1	f1	c1	h1	a1	c1	f1
a2	b2	b2	c3	c2	f3	c2	h1	a2	c3	f3
a3	b2	b3	c2	c3	f3	c3	h1	a3	c3	f3
		b3	c3			c3	h2			

■ **Figure 1** An example of database schema  $\mathbb{R} = \{R_1, R_2, R_3, R_4\}$  (with  $\mathbb{A} = \{A, B, C, F, H\}$ ,  $\text{attr}(R_1) = \{A, B\}$ ,  $\text{attr}(R_2) = \{B, C\}$ ,  $\text{attr}(R_3) = \{C, F\}$  and  $\text{attr}(R_4) = \{C, H\}$ ), a database  $D$ , and the result of CQ  $Q(A, C, F) : -R_1(A, B), R_2(B, C), R_3(C, F), R_4(C, H)$  over  $D$ .  $D' = \{(a_1, b_1), (b_1, c_1), (c_1, f_1), (c_1, h_1)\}$  is the solution to  $\text{SWP}(Q, D, \langle a_1, c_1, f_1 \rangle)$ .  $D'$  together with tuples  $\{(a_2, b_2), (a_3, b_2), (b_2, c_3), (c_3, f_3), (c_3, h_2)\}$  is the solution to  $\text{SWP}(Q, D)$ .

We consider the class of *conjunctive queries without self-joins*:

$$Q(\mathbf{A}) : -R_1(\mathbb{A}_1), R_2(\mathbb{A}_2), \dots, R_m(\mathbb{A}_m)$$

where  $\mathbf{A} \subseteq \mathbb{A}$  is the set of *output attributes* (a.k.a. *free attributes*) and  $\mathbb{A} - \mathbf{A}$  is the set of *non-output attributes* (a.k.a. *existential attributes*). A CQ is *full* if  $\mathbf{A} = \mathbb{A}$ , indicating the natural join among the given relations; otherwise, it is *non-full*. Each  $R_i$  in  $Q$  is distinct. When a CQ  $Q$  is evaluated on database  $D$ , its query result denoted as  $Q(D)$  is the projection of natural join result of  $R_1(\mathbb{A}_1) \bowtie R_2(\mathbb{A}_2) \bowtie \dots \bowtie R_m(\mathbb{A}_m)$  onto  $\mathbf{A}$  (after removing duplicates).

► **Definition 4** (SMALLEST WITNESS PROBLEM (SWP)). *For CQ  $Q$  and database  $D$ , it asks to find a subset of tuples  $D' \subseteq D$  such that  $Q(D) = Q(D')$ , while there exists no subset of tuples  $D'' \subseteq D$  such that  $Q(D'') = Q(D)$  and  $|D''| < |D'|$ .*

Given  $Q$  and  $D$ , we denote the above problem by  $\text{SWP}(Q, D)$ . See an example in Figure 1. We note that the solution to  $\text{SWP}(Q, D)$  may not be unique, hence our target simply finds one such solution. We study the data complexity [38] of  $\text{SWP}$  i.e., the sizes of database schema and query are considered as constants, and the complexity is in terms of input size  $N$ . For any CQ  $Q$  and database  $D$ , the size of query results  $|Q(D)|$  is polynomially large in terms of  $N$ , and  $Q(D)$  can also be computed in polynomial time in terms of  $N$ . In contrast, the size of  $\text{SWP}(Q, D)$  is always smaller than  $N$ , while as we see later  $\text{SWP}(Q, D)$  may not be computed in polynomial time in terms of  $N$ . Again, our target is to compute the smallest witness instead of the query results. We say that  $\text{SWP}$  is *poly-time solvable* for  $Q$  if, for an arbitrary database  $D$ ,  $\text{SWP}(Q, D)$  can be computed in polynomial time in terms of  $|D|$ . As shown later,  $\text{SWP}$  is not poly-time solvable for a large class of CQs, so we introduce an approximated version:

► **Definition 5** ( $\theta$ -APPROXIMATED SMALLEST WITNESS PROBLEM (ASWP)). *For CQ  $Q$  and database  $D$ , it asks to find a subset of tuples  $D' \subseteq D$  such that  $Q(D') = Q(D)$  and  $|D'| \leq \theta \cdot |D^*|$ , where  $D^*$  is a solution to  $\text{SWP}(Q, D)$ .*

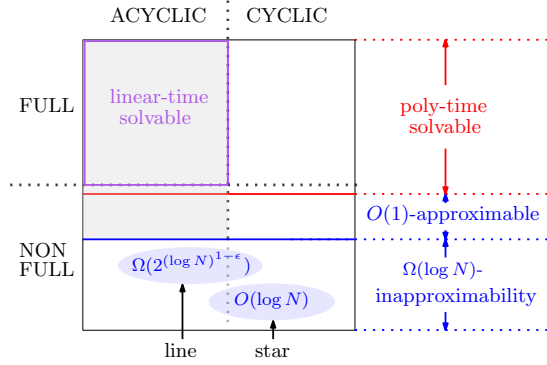
Also,  $\text{SWP}$  is  $\theta$ -approximable for  $Q$  if, for an arbitrary database  $D$ , there is a  $\theta$ -approximated solution to  $\text{SWP}(Q, D)$  that can be computed in polynomial time in terms of  $|D|$ .

## 1.2 Our Results

Our main results obtained can be summarized as (see Figure 2):

In Section 3, we obtain a dichotomy of computing  $\text{SWP}$  for CQs. More specifically,  $\text{SWP}$  for any CQ with head-cluster property (Definition 11) can be solved by a trivial poly-time algorithm, while  $\text{SWP}$  for any CQ without head-cluster property is NP-hard by resorting to the *NP-hardness of set cover* problem (Section 3.2).





■ **Figure 2** Summary of our results. The shadow area is the class of free-connex CQs.

In Section 4, we show a dichotomy of approximating SWP for CQs without head-cluster property. The *head-domination* property that has been identified for *deletion propagation* problem [31], also captures the hardness of approximating SWP. We show a poly-time algorithm that can return a  $O(1)$ -approximated solution to SWP for CQs with head-domination property. On the other hand, we prove that SWP cannot be approximated within a factor of  $(1 - o(1)) \cdot \log N$  for every CQ without head-domination property, unless  $P = NP$ , by resorting to the *logarithmic inapproximability of set cover* problem (in Section 4.2). Interestingly, this separation on approximating SWP for *acyclic* CQs (in Section 2.1) coincides with the separation of *free-connex* and *non-free-connex* CQs (in Section 4.1) in the literature. In Section 5, we further explore approximation algorithms for CQs without head-domination property. Firstly, we show a baseline of returning the union of witnesses for every query result leads to a  $O(N^{1-1/\rho^*})$ -approximated solution, where  $\rho^*$  is the fractional edge covering number<sup>2</sup> of the input CQ [4]. Furthermore, for any CQ with only one non-output attribute, which includes the commonly-studied *star* CQs, we show a greedy algorithm that can approximate SWP within a  $O(\log N)$  factor, matching the lower bound. However, for another commonly-studied class of *line* CQs, which contains more than two non-output attributes, we prove a much higher lower bound  $\Omega(2^{(\log N)^{1-\epsilon}})$  for any  $\epsilon > 0$  in approximating SWP, by resorting to the *label cover* problem (see full version [28]). Meanwhile, we observe that SWP problem for line queries is a special case of the *directed Steiner forest* (DSF) problem (Section 5.3), and therefore existing algorithms for DSF can be applied to SWP directly. But, how to close the gap between the upper and lower bounds on approximating SWP for line CQs remains open.

## 2 Preliminaries

### 2.1 Notations and Classifications of CQs

Extending the notation in Section 1.1, we use  $\mathbf{rels}(Q)$  to denote all the relations that appear in the body of  $Q$ , and use  $\mathbf{attr}(Q), \mathbf{head}(Q) \subseteq \mathbf{attr}(Q)$  to denote all the attributes that appear in the body, head of  $Q$  separately (so,  $\mathbf{head}(Q) = \mathbf{A}$  in Section 1.1). Moreover,  $\mathbf{head}(R_i) = \mathbf{head}(Q) \cap \mathbf{attr}(R_i)$ . For any attribute  $A \in \mathbf{attr}(R_i)$ ,  $\pi_A t$  denotes the value over attribute  $A$  of tuple  $t$ . Similarly, for a set of attributes  $X \subseteq \mathbf{attr}(R_i)$ ,  $\pi_X t$  denotes values over attributes in  $X$  of tuple  $t$ . We also mention two important classes of CQs.

<sup>2</sup> For a CQ  $Q$ , a fractional edge covering is a function  $W : \mathbf{rels}(Q) \rightarrow [0, 1]$  with  $\sum_{R_i: A \in \mathbf{attr}(R_i)} W(R_i) \geq 1$  for every  $A \in \mathbf{A}$ . The fractional edge covering number is the minimum value of  $\sum_{R_i: R_i \in \mathbf{rels}(Q)} W(R_i)$  over all fractional edge coverings.

► **Definition 6** (Acyclic CQs [6, 19]). A CQ  $Q$  is acyclic if there exists a tree  $\mathbb{T}$  such that (1) there is a one-to-one correspondence between the nodes of  $\mathbb{T}$  and relations in  $Q$ ; and (2) for every attribute  $A \in \text{attr}(Q)$ , the set of nodes containing  $A$  forms a connected subtree of  $\mathbb{T}$ . Such a tree is called the join tree of  $Q$ .

► **Definition 7** (Free-connex CQs [5]). A CQ  $Q$  is free-connex if  $Q$  is acyclic and the resulted CQ by adding another relation contains exactly  $\text{head}(Q)$  to  $Q$  is also acyclic.

## 2.2 SWP for One Query Result

The SWP problem for one query result is formally defined as:

► **Definition 8** (SWP FOR ONE QUERY RESULT). For CQ  $Q$ , database  $D$  and query result  $t \in Q(D)$ , it asks for finding a subset of tuples  $D' \subseteq D$  such that  $t \in Q(D')$ , while there is no subset  $D'' \subseteq D$  such that  $t \in Q(D'')$  and  $|D''| < |D'|$ .

Given  $Q$ ,  $D$  and  $t$ , we denote the above problem by  $\text{SWP}(Q, D, t)$ . It has been shown by [32] that  $\text{SWP}(Q, D, t)$  can be computed in polynomial time for arbitrary  $Q$ ,  $D$ , and  $t \in Q(D)$ . Their algorithm [32] simply finds an arbitrary full join result  $t' \in \bowtie_{R_i \in \text{rels}(Q)} R_i$  such that  $\pi_{\text{head}(Q)} t' = t$ , and returns all participating tuples in  $\{\pi_{\text{attr}(R_i)} t' : R_i \in \text{rels}(Q)\}$  as the smallest witness for  $t$ . This primitive is used in building our SWP algorithm. The SWP problem for a Boolean CQ ( $\mathbf{A} = \emptyset$ , indicating whether the result of underlying natural join is empty or not) can be solved by finding SWP for an arbitrary join result in its full version.

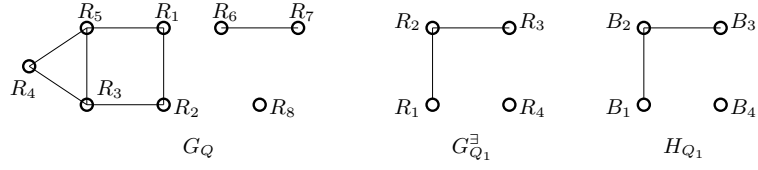
## 2.3 Notions of Connectivity

We give three important notions of connectivity for CQs, which will play an important role in characterizing the structural properties used in SWP. See an example in Figure 3.

**Connectivity of CQ.** We capture the *connectivity* of a CQ  $Q$  by modeling it as a graph  $G_Q$ , where each relation  $R_i$  is a vertex and there is an edge between  $R_i, R_j \in \text{rels}(Q)$  if  $\text{attr}(R_i) \cap \text{attr}(R_j) \neq \emptyset$ . A CQ  $Q$  is *connected* if  $G_Q$  is connected, and *disconnected* otherwise. For a disconnected CQ  $Q$ , we can decompose it into multiple connected subqueries by applying search algorithms on  $G_Q$ , and finding all connected components for  $G_Q$ . The set of relations corresponding to the set of vertices in one connected component of  $G_Q$  form a connected subquery of  $Q$ . Given a disconnected CQ  $Q$ , let  $Q_1, Q_2, \dots, Q_k$  be its connected subqueries. Given a database  $D$  over  $Q$ , let  $D_i \subseteq D$  be the corresponding sub-databases defined for  $Q_i$ . Observe that every witness for  $Q(D)$  is the disjoint union of a witness for  $Q_i(D_i)$ , for  $i \in [k]$ . Hence, Lemma 9 follows. In the remaining of this paper, we assume that  $Q$  is connected.

► **Lemma 9.** For a disconnected CQ  $Q$  of  $k$  connected components  $Q_1, Q_2, \dots, Q_k$ , SWP is poly-time solvable for  $Q$  if and only if SWP is poly-time solvable for every  $Q_i$ , where  $i \in [k]$ .

**Existential-Connectivity of CQ.** We capture the existential-connectivity of a CQ  $Q$  by modeling it as a graph  $G_Q^\exists$ , where each relation  $R_i$  with  $\text{attr}(R_i) - \text{head}(Q) \neq \emptyset$  is a vertex, and there is an edge between  $R_i, R_j \in \text{rels}(Q)$  if  $\text{attr}(R_i) \cap \text{attr}(R_j) - \text{head}(Q) \neq \emptyset$ . We can find the connected components of  $G_Q^\exists$  by applying search algorithm on  $G_Q^\exists$ , and finding all connected components for  $G_Q^\exists$ . Let  $E_1, E_2, \dots, E_k \subseteq \text{rels}(Q)$  be the connected components of  $G_Q^\exists$ , each corresponding to a subset of relations in  $Q$ .



■ **Figure 3** An illustration of  $G_Q$  for  $Q(A_1, A_2, A_3, A_4, A_5) : - R_1(A_1, B_1), R_2(B_1, B_2), R_3(A_2, B_2, B_3), R_4(A_2, A_3, B_4), R_5(A_1, A_2), R_6(A_4, B_5), R_7(B_5, A_5), R_8(B_6, B_7)$  with three subqueries  $Q_1(A_1, A_2, A_3) : - R_1(A_1, B_1), R_2(B_1, B_2), R_3(A_2, B_2, B_3), R_4(A_2, A_3, B_4), R_5(A_1, A_2)$ , and  $Q_2(A_4, A_5) : - R_6(A_4, B_5), R_7(B_5, A_5)$  and  $Q_3 : - R_8(B_6, B_7)$ . The middle is  $G_{Q_1}^=$  for  $Q_1$ , with two connected components  $\{R_1, R_2, R_3\}, \{R_4\}$  and dominants  $R_5, R_4$ . The right is  $H_{Q_1}$  for  $Q_1$ , with two connected components  $\{B_1, B_2, B_3\}, \{B_4\}$ .

**Nonout-Connectivity of CQ.** We capture the nonout-connectivity of a CQ  $Q$  by modeling it as a graph  $H_Q$ , where each non-output attribute  $A \in \text{attr}(Q) - \text{head}(Q)$  is a vertex, and there is an edge between  $A, B \in \text{attr}(Q) - \text{head}(Q)$  if there exists a relation  $R_i \in \text{rels}(Q)$  such that  $A, B \in \text{attr}(R_i)$ . We can find the connected components of  $H_Q$ , and finding all connected components for  $H_Q$ . Let  $H_1, H_2, \dots, H_k \subseteq \text{attr}(Q) - \text{head}(Q)$  be the connected components of  $H_Q$ , each corresponding to a subset of non-output attributes in  $Q$ .

## 2.4 Head Cluster and Domination

These two important structural properties identified for characterizing the hardness of (A)SWP are directly built on the existential-connectivity of a CQ  $Q$  and the notion of dominant relation. For a CQ  $Q$  with a subset  $E \subseteq \text{rels}(Q)$  of relations,  $R_i \in \text{rels}(Q)$  is a *dominant* relation for  $E$  if every output attribute appearing in any relation of  $E$  also appears in  $R_i$ , i.e.,  $\cup_{R_j \in E} \text{head}(R_j) \subseteq \text{head}(R_i)$ .

► **Definition 10** (Head Domination [30]). *For CQ  $Q$ , let  $E_1, E_2, \dots, E_k$  be the connected components of  $G_Q^=$ .  $Q$  has head-domination property if for any  $i \in [k]$ , there exists a dominant relation from  $\text{rels}(Q)$  for  $E_i$ .*

The notion of head-domination property has been first identified for deletion propagation with side effect problem [31], which studied the smallest number of tuples to remove so that a subset of desired query results must disappear while maintain as many as remaining query results. We give a detailed comparison between SWP and deletion propagation in the full version [28], although they solve completely independent problem for CQs without self-joins.

► **Definition 11** (Head Cluster). *For CQ  $Q$ , let  $E_1, \dots, E_k$  be connected components of  $G_Q^=$ .  $Q$  has head-cluster property if for any  $i \in [k]$ , every  $R_j \in E_i$  is a dominant relation for  $E_i$ .*

There is an equivalent but simpler definition for head-cluster property: A CQ  $Q$  has head-cluster property if for every pair of relations  $R_i, R_j \in \text{rels}(Q)$  with  $\text{head}(R_i) \neq \text{head}(R_j)$ , there must be  $\text{attr}(R_i) \cap \text{attr}(R_j) \subseteq \text{head}(Q)$ . Here, we define head-cluster property based on dominant relation, since it is a special case of head-domination property.

## 3 Dichotomy of Exact SWP

In this section, we focus on computing SWP exactly for CQs, which can be efficiently done if *head-cluster* property is satisfied. All missing proofs are given in the full version [28].

► **Theorem 12.** *If a CQ  $Q$  has head-cluster property, SWP is poly-time solvable; otherwise, SWP is not poly-time solvable, unless  $P = NP$ .*

---

**Algorithm 1**  $\text{SWP}(Q, D)$ .

---

```

1  $D' \leftarrow \emptyset$ ;
2  $(E_1, E_2, \dots, E_k) \leftarrow$  connected components of  $G_Q^\exists$ ;
3  $\mathbf{A}_1, \mathbf{A}_2, \dots, \mathbf{A}_k \leftarrow$  output attributes of  $E_1, E_2, \dots, E_k$ ;
4 foreach  $R_j \in \text{rels}(Q)$  with  $\text{attr}(R_j) \subseteq \text{head}(Q)$  do
5    $D' \leftarrow D' \cup \pi_{\text{attr}(R_j)} Q(D)$ ;
6 foreach  $i \in [k]$  do
7   Define  $Q_i(\mathbf{A}_i) : -\{R_j(\mathbf{A}_j) : R_j \in E_i\}$ ;
8   foreach  $t' \in \pi_{\mathbf{A}_i} Q(D)$  do
9      $D' \leftarrow D' \uplus \text{SWP}(Q_i, \{R_j : R_j \in E_i\}, t')$ ;
10 return  $D'$ ;

```

---

### 3.1 An Exact Algorithm

We prove the first part of Theorem 12 with a poly-time algorithm. The head-cluster property implies that if two relations have different output attributes, they share no common non-output attributes. This way, we can cluster relations by output attributes. As shown, Algorithm 1 partitions all relations into  $\{E_1, E_2, \dots, E_k\}$  based on the connected components in  $G_Q^\exists$ . For the subset of relations in one connected component  $E_i$ , every relation is a dominant relation, i.e., shares the same output attributes. If one relation only contains output attributes  $\mathbf{A}_i$  (line 4), it must appear alone as a singleton component, since we assume there is no duplicate relations in the input CQ. All tuples from such a relation that participate in any query results must be included by every witness to  $Q(D)$ . We next consider the remaining components containing at least two relations. In  $E_i$ , for each tuple  $t' \in \pi_{\mathbf{A}_i} Q(D)$  in the projection of query results onto the output attributes  $\mathbf{A}_i$ , Algorithm 1 computes the smallest witness for  $t'$  in sub-query  $Q_i$  defined on relations in  $E_i$ . The disjoint union ( $\uplus$ ) of witnesses returned for all groups forms the final witness. On a CQ with head-cluster property, Algorithm 1 can be stated in a simpler way (see the full version [28]). Algorithm 1 runs in polynomial time, as (i)  $Q(D)$  can be computed in polynomial time; (ii)  $|Q(D)|$  is polynomially large; and (iii) the primitive in line 9 only takes  $O(1)$  time.

► **Lemma 13.** *For a CQ  $Q$  with head-cluster property, Algorithm 1 finds a solution to  $\text{SWP}(Q, D)$  for any database  $D$  in polynomial time.*

**Proof.** We prove that for any CQ  $Q$  with head-cluster property and an arbitrary database  $D$ , Algorithm 1 returns the solution to  $\text{SWP}(Q, D)$ . Together with the fact that Algorithm 1 runs in polynomial time, we finish the proof for Lemma 13. Let  $D'$  be the solution returned by Algorithm 1. Let  $D'_i \subseteq D'$  be the set of tuples from relations in  $E_i$ . We show that  $D'$  is a witness for  $Q(D)$ , i.e.,  $Q(D') = Q(D)$ .

**Direction  $\subseteq$ .** As  $Q$  is monotone,  $Q(D') \subseteq Q(D)$  holds for any sub-database  $D' \subseteq D$ .

**Direction  $\supseteq$ .** Consider an arbitrary query result  $t \in Q(D)$ . Let  $D'_i(t)$  denote the group of tuples returned by  $\text{SWP}(Q_i, \{R_j : R_j \in E_i\}, \pi_{\mathbf{A}_i} t)$ . We note that  $t \in \pi_{\mathbf{A}} \bowtie_{i \in [k]} D'_i(t)$ , since

- every tuple has the same value  $\pi_A t$  over any output attribute  $A$ , if it contains attribute  $A$ ;
- there is no non-output attribute to join for tuples across groups;
- tuples inside each group can be joined by non-output attribute; (implied by the correctness of  $\text{SWP}$  for a single query result)

Hence,  $t \in Q(D')$ . Together,  $Q(D') \supseteq Q(D)$ .

We next show that there exists no  $D'' \subseteq D$  such that  $Q(D'') = Q(D)$  and  $|D''| < |D'|$ . Suppose not, let  $D''_i \subseteq D''$  denote the set of tuples from relations in  $E_i$ . As  $|D''| < |D'|$ , there must exist some  $i \in [k]$  such that  $|D''_i| < |D'_i|$ , i.e.,  $D'_i - D''_i \neq \emptyset$ . In Algorithm 1, we can rewrite  $D'_i$  as follows:

$$D'_i = \bigsqcup_{t' \in \pi_{\mathbf{A}_i} Q(D)} \text{SWP}(Q_i, \{R_i : R_i \in E_i\}, t'),$$

where  $\bigsqcup$  denotes the disjoint union. As  $D'_i - D''_i \neq \emptyset$ , there must exist some  $t' \in \pi_{\mathbf{A}_i} Q(D)$  such that  $t' \notin Q_i(D''_i)$ , i.e.,  $t'$  cannot be witnessed by  $D''_i$ . Correspondingly, there must exist some query result  $t$  with  $\pi_{\mathbf{A}_i} t = t'$  such that  $t \notin Q(D'')$ , i.e.  $t$  cannot be witnessed by  $D''$ , contradicting the fact that  $Q(D'') = Q(D)$ . Hence, no such  $D''$  exists.  $\blacktriangleleft$

► **Remark 1.** SWP is poly-time solvable for any full CQ, since  $\text{attr}(R_i) \cap \text{attr}(R_j) \subseteq \text{head}(Q)$  holds for every pair of relations  $R_i, R_j \in \text{rels}(Q)$ . Hence, the hardness of SWP comes from projection. On the other hand, SWP is also poly-time solvable for some non-full CQs, say  $Q(A_1, A_2, A_3) : -R_1(A_1, A_2), R_2(A_2, A_3), R_3(A_1, A_3), R_4(A_1, B_1)$ .

► **Remark 2.** It is not always necessary to compute  $Q(D)$  as Algorithm 1 does. We actually have much faster algorithms for some classes of CQs. If CQ  $Q$  is full,  $\text{SWP}(Q, D)$  is the set of *non-dangling* tuples in  $D$ , i.e., those participate in at least one query result of  $Q(D)$ . Furthermore, if  $Q$  is an acyclic full CQ, non-dangling tuples can be identified in  $O(|D|)$  time [40]. It is more expensive to identify non-dangling tuples for cyclic full CQs, for example, the PANDA algorithm [2] can identify non-dangling tuples for any full CQ in  $O(N^w)$  time, where  $w \geq 1$  is the sub-modular width of input query [2]. It is left as an interesting open question to compute SWP for CQs with head-cluster property more efficiently.

### 3.2 Hardness

We next prove the second part of Theorem 12 by showing the hardness for CQs without head-cluster property. Our hardness result is built on the NP-hardness of set cover [7]: Given a universe  $\mathcal{U}$  of  $n$  elements and a family  $\mathcal{S}$  of subsets of  $\mathcal{U}$ , it asks to find a subfamily  $\mathcal{C} \subseteq \mathcal{S}$  of sets whose union is  $\mathcal{U}$  (called “cover”), while using the fewest sets. We start with  $Q_{\text{cover}}(A) : -R_1(A, B), R_2(B)$  and then extend to any CQ without head-cluster property.

► **Lemma 14.** *SWP is not poly-time solvable for  $Q_{\text{cover}}(A) : -R_1(A, B), R_2(B)$ , unless  $P = NP$ .*

**Proof.** We show a reduction from set cover to SWP for  $Q_{\text{cover}}$ . Given an arbitrary instance  $(\mathcal{U}, \mathcal{S})$  of set cover, we construct a database  $D$  for  $Q_{\text{cover}}$  as follows. For each element  $u \in \mathcal{U}$ , we add a value  $a_u$  to  $\text{dom}(A)$ ; for each subset  $S \in \mathcal{S}$ , we add a value  $b_S$  to  $\text{dom}(B)$ , and a tuple  $(b_S)$  to  $R_2$ . Moreover, for each pair  $(u, S) \in \mathcal{U} \times \mathcal{S}$  with  $u \in S$ , we add a tuple  $(a_u, b_S)$  to  $R_1$ . Note that  $Q_{\text{cover}}(D) = \mathcal{U}$ . It is now to show that  $(\mathcal{U}, \mathcal{S})$  has a cover of size  $\leq k$  if and only if  $\text{SWP}(Q_{\text{cover}}, D)$  has a solution  $D'$  of size  $\leq |\mathcal{U}| + k$ . Then, if SWP is poly-time solvable for  $Q_{\text{cover}}$ , set cover is also poly-time solvable, which is impossible unless  $P = NP$ .  $\blacktriangleleft$

► **Lemma 15.** *For a CQ  $Q$  without head-cluster property, SWP is not poly-time solvable for  $Q$ , unless  $P = NP$ .*

**Proof.** Consider such a CQ  $Q$  with a desired pair of relations  $R_i, R_j \in \text{rels}(Q)$ . We next show a reduction from  $Q_{\text{cover}}$  to  $Q$ . Given an arbitrary database  $D_{\text{cover}}$  over  $Q_{\text{cover}}$ , we construct a database  $D$  over  $Q$  as follows. First, it is always feasible to identify attribute  $A' \in \text{head}(R_i) - \text{attr}(R_j)$  and attribute  $B' \in \text{attr}(R_i) \cap \text{attr}(R_j) - \text{head}(Q)$ . We set  $\text{dom}(A') = \text{dom}(A)$ ,  $\text{dom}(B') = \text{dom}(B)$ , and remaining attributes with a dummy value  $\{*\}$ .

Each relation in  $Q$  degenerates to  $R_1(A, B)$ , or  $R_2(B)$ , or a dummy tuple  $\{*, *, \dots, *\}$ . It can be easily checked that there is a one-to-one correspondence between solutions to  $\text{SWP}(Q, D)$  and solutions to  $\text{SWP}(Q_{\text{cover}}, D_{\text{cover}})$ . Thus, if  $\text{SWP}$  is poly-time solvable for  $Q$ , then  $\text{SWP}$  is also poly-time solvable for  $Q_{\text{cover}}$ , coming to a contradiction of Lemma 14.  $\blacktriangleleft$

## 4 Dichotomy of Approximated SWP

As it is inherently difficult to compute  $\text{SWP}$  exactly for general CQs, the next interesting question is to explore approximated solutions for  $\text{SWP}$ . In this section, we establish the following dichotomy for approximating  $\text{SWP}$ . All missing proofs are given in the full version [28].

► **Theorem 16.** *If a CQ  $Q$  has head-domination property,  $\text{SWP}$  is  $O(1)$ -approximable; otherwise,  $\text{SWP}$  of input size  $N$  is not  $(1 - o(1)) \cdot \log N$ -approximable, unless  $P = NP$ .*

### 4.1 A $O(1)$ -Approximation Algorithm

Let's start by revisiting  $Q_{\text{cover}}(A) : -R_1(A, B), R_2(B)$ . Although  $\text{SWP}$  is hard to compute exactly for  $Q_{\text{cover}}$ , it is easy to approximate  $\text{SWP}(Q_{\text{cover}}, D)$  for arbitrary database  $D$  within a factor of 2. Let  $D^*$  be a solution to  $\text{SWP}(Q_{\text{cover}}, D)$ . We can simply construct an approximated solution  $D'$  by picking a pair of tuples  $(a, b) \in R_1, (b) \in R_2$  for every  $a \in Q_{\text{cover}}(D)$ , and show that  $|D'| \leq 2 \cdot |Q_{\text{cover}}(D)| \leq 2 \cdot |D^*|$ . This is actually not a violation to the inapproximability of set cover problem. If revisiting the proof of Lemma 14,  $(\mathcal{U}, \mathcal{S})$  has a cover of size  $\leq k$  if and only if  $\text{SWP}(Q_{\text{cover}}, D)$  has a solution of size  $\leq |\mathcal{U}| + k$ . Due to the fact that  $k \leq |\mathcal{U}|$ , the inapproximability of set cover does not carry over to  $\text{SWP}$  for  $Q$ . This observation can be generalized to all CQs with head-domination property.

As described in Algorithm 1, an approximated solution to  $\text{SWP}(Q, D)$  for a CQ  $Q$  with head-domination property consists of two parts. For every relation that only contains output attributes, Algorithm 1 includes all tuples that participate in at least one query result (line 4–5), which must be included by any witness for  $Q(D)$ . For the remaining relations, Algorithm 1 partitions them into groups based on the existential connectivity. Intuitively, every pair of relations across groups can only join via output attributes in their dominants. Recall that  $\mathbf{A}_i$  denotes the set of output attributes appearing in relations from  $E_i$ . Then, for each group  $E_i$ , we consider each tuple  $t' \in \pi_{\mathbf{A}_i} Q(D)$  and find the smallest witness for  $t'$  in  $Q_i$  defined by relations in  $E_i$ . The union of witnesses returned for all groups forms the final answer. As shown before, Algorithm 1 runs in polynomial time.

► **Lemma 17.** *For a CQ  $Q$  with head-domination property, Algorithm 1 finds a  $O(1)$ -approximated solution to  $\text{SWP}(Q, D)$  for any database  $D$  in polynomial time.*

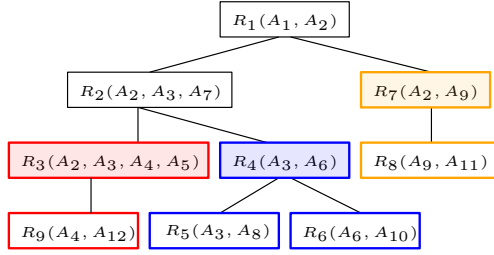
**Proof.** Consider the connected components  $E_1, E_2, \dots, E_k$  of  $G_Q^\exists$  with dominants  $\dot{R}_1, \dot{R}_2, \dots, \dot{R}_k$ . Let  $Q_i$  be the subquery defined over relations in  $E_i$ , with output attributes  $\text{head}(Q_i) = \text{attr}(\dot{R}_i)$ , and  $D_i = \{R_j : R_j \in E_i\}$  be the corresponding database for  $Q_i$ . Let  $D^*$  be the solution to  $\text{SWP}(Q, D)$ . We point out some observations on  $D^*$ :

- For each  $R_j$  with  $\text{attr}(R_j) \subseteq \text{head}(Q)$ ,  $D^*$  must include all tuples in  $\pi_{\text{attr}(R_j)} Q(D)$ .
- For every dominant  $\dot{R}_i$ ,  $D^*$  must include at least  $|\pi_{\text{head}(\dot{R}_i)} Q(D)|$  tuples from  $\dot{R}_i$ .

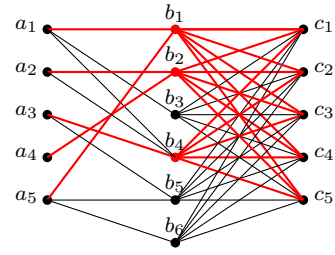
On the other hand, Algorithm 1 includes  $|\text{rels}(Q_i)|$  tuples for each primitive at line 11, and invokes this primitive for each tuple in  $\pi_{\text{head}(\dot{R}_i)} Q(D)$ . Together, we come to:

$$\begin{aligned} |D'| &= \sum_{R_j: \text{attr}(R_j) \subseteq \text{head}(Q)} |\pi_{\text{attr}(R_j)} Q(D)| + \sum_{\dot{R}_i: i \in [k]} |\pi_{\text{head}(\dot{R}_i)} Q(D)| \cdot |\text{rels}(Q_i)| \\ &\leq |D^*| + |D^*| \cdot |\text{rels}(Q)| \leq 2 \cdot |D^*| \cdot |\text{rels}(Q)| \end{aligned}$$





■ **Figure 4** A free-connex CQ  $Q$  with  $\text{head}(Q) = \{A_1, A_2, A_3, A_7\}$ . A partition of relations containing non-output attributes is  $\{\{R_3, R_9\}, \{R_4, R_5, R_6\}, \{R_7, R_8\}\}$ , with dominant relations  $R_3, R_4, R_7$  respectively.



■ **Figure 5** A database  $D$  for  $Q_{\text{matrix}}$  with an integral sub-database in red. Each vertex is a value in the attribute, and each edge is a tuple in  $D$ .

It can also be easily checked that  $Q(D) = Q(D')$ . Hence, Algorithm 1 always returns a  $O(1)$ -approximation solution for  $\text{SWP}(Q, D)$ . Moreover, the query results  $Q(D)$  can be computed in poly-time in the input size of  $D$ . Each primitive of finding the smallest witness for one query result takes  $O(1)$  time. Hence, Algorithm 1 runs in polynomial time. ◀

**Connection with Free-connex CQs.** We point out that every free-connex CQ has head-domination property, which is built on an important property as stated in Lemma 18.

► **Lemma 18** ([5]). *A free-connex CQ has a tree structure  $\mathbb{T}$  such that (1) each node of  $\mathbb{T}$  corresponds to  $\text{attr}(R_i)$  or  $\text{head}(R_i)$  for some relation  $R_i \in \text{rels}(Q)$ ; and for each relation  $R_i \in \text{rels}(Q)$ , there exists a node of  $\mathbb{T}$  corresponding to  $\text{attr}(R_i)$ ; (2) for every attribute  $A \in \text{attr}(Q)$ , the set of nodes containing  $A$  form a connected subtree of  $\mathbb{T}$ ; (3) there is a connected subtree  $\mathbb{T}_{\text{con}}$  of  $\mathbb{T}$  including the root of  $\mathbb{T}$ , such that the set of attributes appearing in  $\mathbb{T}_{\text{con}}$  is exactly  $\text{head}(Q)$ .*

► **Lemma 19.** *Every free-connex CQ has head-domination property.*

**Proof.** Recall that  $Q$  has head-domination property if for any connected component  $E_j$  of  $G_Q^\exists$ , there exists a dominant relation from  $\text{rels}(Q)$  for  $E_j$ . Suppose we are given a tree structure  $\mathbb{T}$  for a free-connex CQ  $Q$  with  $\mathbb{T}_{\text{con}}$  as described by Lemma 18. Consider an arbitrary relation  $R_i$  with  $\text{attr}(R_i) - \text{head}(R_i) \neq \emptyset$ , such that all its ancestors in  $\mathbb{T}$  only contain output attributes. Let  $\mathbb{T}_i$  be the subtree rooted at  $R_i$ . We note that any relation in  $\mathbb{T}_i$  cannot fall into the same connected component with a relation in  $\mathbb{T} \setminus \mathbb{T}_i$ , since they do not share any common non-output attribute. Consider any relation  $R_j \in \mathbb{T}_i$ . Implied by the fact that  $\mathbb{T}_{\text{con}}$  is a connected subtree and  $R_i \notin \mathbb{T}_{\text{con}}$ , we have  $R_j \notin \mathbb{T}_{\text{con}}$ . Then,  $\text{head}(R_j) \subseteq \text{head}(R_i)$ ; otherwise, any output attribute in  $\text{head}(R_j) - \text{head}(R_i)$  does not appear in  $\mathbb{T}_{\text{con}}$ . Hence, for any connected component  $E_j$  formed by relations from  $\mathbb{T}_i$ ,  $R_i$  is a dominant relation for  $E_j$ . This argument applies for every connected component of  $G_Q^\exists$ . ◀

## 4.2 Logarithmic Inapproximability

Now, we turn to the class of CQs without head-domination property, and show their hardness by resorting to inapproximability of set cover [20, 17]: there is no poly-time algorithm for approximating set cover of input size  $n$  within factor  $(1 - o(1)) \cdot \log n$ , unless  $\text{P} = \text{NP}$ . We identify two hardcore structures (Definition 23 and Definition 26), and prove that no poly-time algorithm can approximate  $\text{SWP}$  for any CQ containing a hardcore within a logarithmic factor, unless  $\text{P} = \text{NP}$ . Lastly, we complete the proof of Theorem 16 by establishing the connection between the non-existence of a hardcore and head-domination property for CQs.



$$\begin{array}{ll}
(O1) & \min \sum_{b \in B} |C_b| \\
& \text{s.t. } \bigcup_{b:(a,b) \in R_1} C_b = C, \forall a \in A \\
& \quad C_b \subseteq C, \forall b \in B
\end{array}
\qquad
\begin{array}{ll}
(O2) & \min \sum_{b \in B} x_b \\
& \text{s.t. } \sum_{b:(a,b) \in R_1} x_b \geq 1, \forall a \in A \\
& \quad x_b \in \{0, 1\}, \forall b \in B
\end{array}$$

■ **Figure 6** Optimization problems in the proof of Lemma 21.

### 4.2.1 Free sequence

Let's start with the simplest acyclic but non-free-connex CQ  $Q_{\text{matrix}}(A, C) : -R_1(A, B), R_2(B, C)$ . We show a reduction from set cover to SWP for  $Q_{\text{matrix}}$  while preserving its logarithmic inapproximability. The essence of is the notion of *integral witness*, such that for any database  $D$  where  $R_2$  is a Cartesian product,  $\text{SWP}(Q_{\text{matrix}}, D)$  always admits an integral witness.

► **Definition 20** (Integral Database). *For any database  $D$  over  $Q_{\text{matrix}}$  with  $R_2 = (\pi_B R_2) \times (\pi_C R_2)$ , a sub-database  $(R'_1, R'_2) \subseteq D$  is integral if  $R'_2 = (\pi_B R'_2) \times (\pi_C R'_2)$ .*

► **Lemma 21.** *For  $Q_{\text{matrix}}$  and any database  $D$  where  $R_2 = (\pi_B R_2) \times (\pi_C R_2)$ , there is an integral solution to  $\text{SWP}(Q_{\text{matrix}}, D)$ , i.e., a smallest witness to  $Q(D)$  that is also integral.*

**Proof.** Consider a database  $D$  where  $R_2 = (\pi_B R_2) \times (\pi_C R_2)$ . We assume that there exists no dangling tuples in  $R_1, R_2$ , i.e., every tuple can join with some tuple from the other relation; otherwise, we simply remove these dangling tuples. With a slight abuse of notation, we denote  $A = \pi_A R_1$ ,  $B = \pi_B R_1 (= \pi_B R_2)$  and  $C = \pi_C R_2$ . We consider the optimization problem  $O1$  in Figure 6. Intuitively, it asks to assign a subset of elements  $C_b \subseteq C$  to each value  $b \in B$ , such that each  $a$  is “connected” to all values in  $C$  via tuples in  $R_1, R_2$ , while the total size of the assignment defined as  $\sum_{b \in B} |C_b|$  is minimized. See Figure 5.

We rewrite the objective function as:  $\sum_{b \in B} |C_b| = \sum_{c \in C} |\{b \in B : c \in C_b\}|$ , where  $\{b \in B : c \in C_b\}$  indicates the subset of values from  $B$  to which  $c$  is assigned. Together with the constraint that every  $a \in A$  must be connected to  $c$ , we note that minimizing  $|\{b \in B : c \in C_b\}|$  is equivalent to solving the optimization problem  $(O2)$  in Figure 6. Let  $x^*$  be the optimal solution of the program above, which only depends on the input relation  $R_1$ , and completely independent of the specific value  $c$ . Hence, we conclude that  $\sum_{b \in B} |C_b| = |C| \cdot \sum_{b \in B} x_b^*$ .

We next construct an integral sub-database  $D'$  as follows. For each  $b \in B$  with  $x_b^* = 1$ , we add tuples in  $\{(b, c) \in R_2 : \forall c \in C\}$  to  $D'$ . For each  $a \in A$ , we pick an arbitrary  $b \in B$  with  $(a, b) \in R_1$  and  $x_b^* = 1$ , and add  $(a, b)$  to  $D'$ . Any solution to  $\text{SWP}(Q_{\text{matrix}}, D)$  must contain at least  $|A|$  tuples from  $R_1$  and at least  $|C| \cdot \sum_{b \in B} x_b^*$  tuples from  $R_2$ . Hence,  $D'$  is an integral witness to  $\text{SWP}(Q_{\text{matrix}}, D)$ . ◀

► **Lemma 22.** *There is no poly-time algorithm to approximate SWP for  $Q_{\text{matrix}}$  within a factor of  $(1 - o(1)) \cdot \log N$ , unless  $P = NP$ .*

**Proof.** Consider an arbitrary instance of set cover  $(\mathcal{U}, \mathcal{S})$ , where  $|\mathcal{U}| = n$  and  $|\mathcal{S}| = n^c$  for some constant  $c \geq 1$ .<sup>3</sup> We construct a database  $D$  for  $Q_{\text{matrix}}$  as follows. For each element  $u \in \mathcal{U}$ , we add a value  $a_u$  to  $\text{dom}(A)$  and  $c_u$  to  $\text{dom}(C)$ ; for each subset  $S \in \mathcal{S}$ , we add a value  $b_S$  to  $\text{dom}(B)$ . Moreover, for each pair  $(u, S) \in \mathcal{U} \times \mathcal{S}$  with  $u \in S$ , we add tuple  $(a_u, b_S)$

<sup>3</sup> The inapproximability of set cover holds even when the size of the family of subsets is only polynomially large with respect to the size of the universe of elements [33].

## 24:12 Finding Smallest Witnesses for Conjunctive Queries

to  $R_1$ .  $R_2$  is a Cartesian product between  $\text{dom}(B)$  and  $\text{dom}(C)$ . Every relation contains at most  $n^{c+1}$  tuples. Hence  $N \leq 2n^{c+1}$ . Note that  $Q_{\text{matrix}}(D)$  is the Cartesian product between  $A$  and  $C$ . Implied by Lemma 21, it suffices to consider integral witness to  $\text{SWP}(Q_{\text{matrix}}, D)$ . Here, we show that  $(\mathcal{U}, \mathcal{S})$  has a cover of size  $\leq k$  if and only if the  $\text{SWP}(Q_{\text{matrix}}, D)$  has an integral solution of size  $\leq |\mathcal{U}| + k \cdot |V| = n(k+1)$ .

**Direction only-if.** Suppose we are given a cover  $\mathcal{S}'$  of size  $k$  to  $(\mathcal{U}, \mathcal{S})$ . We construct an integral solution  $D'$  to  $\text{SWP}(Q_{\text{matrix}}, D)$  as follows. Let  $B' \subseteq \text{dom}(B)$  be the set of values that corresponding to  $\mathcal{S}'$ . For every  $b_S \in B'$ , i.e.,  $S \in \mathcal{S}'$ , we add tuple  $(b_S, c_u)$  to  $D'$  for every  $u \in \mathcal{U}$ . For every  $a_u \in \text{dom}(A)$ , we choose an arbitrary value  $b_S \in B'$  such that  $u \in S$ , and add tuple  $(a_u, b_S)$  to  $D'$ . This is always feasible since  $\mathcal{S}'$  is a valid set cover. It can be easily checked that  $n$  tuples from  $R_1$  and  $k \cdot n$  tuples from  $R_2$  are added to  $D'$ .

**Direction if.** Suppose we are given an integral solution  $D'$  of size  $k'$  to  $\text{SWP}(Q_{\text{matrix}}, D)$ . Let  $B'$  be the subset of values whose incident tuples in  $R_2$  are included by  $D'$ . We argue that all subsets corresponding to  $B'$  forms a valid cover of size  $\frac{k'}{n-1}$ . By definition of integral solution,  $|B'| = \frac{k'}{n} - 1$ . Moreover, for every value  $a \in \text{dom}(A)$ , at least one edge  $(a, b)$  is included by  $D'$  for some  $b \in B'$ , hence  $B'$  must be a valid set cover.

Hence, if  $\text{SWP}$  is  $(1 - o(1)) \cdot \log N$ -approximable for  $Q_{\text{matrix}}$ , there is a poly-time algorithm that approximates set cover of input size  $n$  within a  $(1 - o(1)) \cdot \log N$  factor, which is impossible unless  $P = NP$ .  $\blacktriangleleft$

► **Definition 23 (Free Sequence).** In a CQ  $Q$ , a free sequence is a sequence of attributes  $P = \langle A_1, A_2, \dots, A_k \rangle$  such that<sup>4</sup>

- $A_1, A_k \in \text{head}(Q)$  and  $A_2, A_3, \dots, A_{k-1} \in \text{attr}(Q) - \text{head}(Q)$ ;
- for every  $i \in [k-1]$ , there exists a relation  $R_j \in \text{rels}(Q)$  such that  $A_i, A_{i+1} \in \text{attr}(R_j)$ ;
- there exists no relation  $R_j \in \text{rels}(Q)$  such that  $A_1, A_k \in R_j$ .

► **Lemma 24.** For a CQ  $Q$  containing a free sequence, there is no poly-time algorithm that can approximate  $\text{SWP}$  for  $Q$  within a factor of  $(1 - o(1)) \cdot \log N$ , unless  $P = NP$ .

**Proof.** Let  $P = \langle A_1, A_2, \dots, A_k \rangle$  be such a sequence. For simplicity, let  $P' = \{A_2, A_3, \dots, A_{k-1}\}$ . We next show a reduction from set cover to  $\text{SWP}(Q, D)$ . Consider an arbitrary instance of set cover with a universe  $\mathcal{U}$  and a family  $\mathcal{S}$  of subsets of  $\mathcal{U}$  where  $|\mathcal{U}| = n$  and  $|\mathcal{S}| = n^c$  for some constant  $c \geq 1$ . We construct a database  $D$  for  $Q$  as follows. For each  $u \in \mathcal{U}$ , we add a value  $a_u$  to  $\text{dom}(A_1)$  and  $\text{dom}(A_k)$ . For each subset  $S \in \mathcal{S}$ , we add a value  $b_S$  to  $\text{dom}(B)$  for every  $B \in P'$ . We set the domain of remaining attributes in  $\text{attr}(Q) - P$  as  $\{*\}$ . For each relation  $R_j \in \text{rels}(Q)$ , we distinguish the following cases:

- If  $A_1 \in \text{attr}(R_j)$ , we further distinguish two more cases:
  - if  $\text{attr}(R_j) \cap P' = \emptyset$ , we add tuple  $t$  with  $\pi_{A_1} t = a_u$  for every  $u \in \mathcal{U}$ ;
  - otherwise, we add tuple  $t$  with  $\pi_{A_1} t = a_u$  and  $\pi_{A_i} t = b_S$  for  $A_i \in \text{attr}(R_j) \cap P'$ , for every pair  $(u, S) \in \mathcal{U} \times \mathcal{S}$  such that  $u \in S$ ;
- If  $A_k \in \text{attr}(R_j)$ , we further distinguish two more cases:
  - if  $\text{attr}(R_j) \cap P' = \emptyset$ , we add tuple  $t$  with  $\pi_{A_k} t = a_u$  for every  $u \in \mathcal{U}$ ;
  - otherwise, we add tuple  $t$  with  $\pi_{A_k} t = a_u$  and  $\pi_{A_i} t = b_S$  for  $A_i \in \text{attr}(R_j) \cap P'$ , for every pair  $(u, S) \in \mathcal{U} \times \mathcal{S}$ ;
- If  $P \cap \text{attr}(R_j) \subseteq P - \{A_1, A_k\}$ , we add a tuple  $t$  with  $\pi_{A_i} t = b_S$  for  $A_i \in \text{attr}(R_j) \cap P$ , for every  $S \in \mathcal{S}$ ;
- If  $P \cap \text{attr}(R_j) = \emptyset$ , then we add a tuple  $\{*\}$ ;

<sup>4</sup> Free sequence is a slight generalized notion of free path [5] studied in the literature, which further requires that for any relation  $R_j \in \text{rels}(Q)$ , either  $\text{attr}(R_j) \cap P = \emptyset$ , or  $|\text{attr}(R_j) \cap P| = 1$ , or  $\text{attr}(R_j) \cap P = \{A_i, A_{i+1}\}$  for some  $i \in [k-1]$ .

It can be easily checked that every relation contains at most  $n^{c+1}$  tuples, hence  $\log N = \Theta(\log n)$ . The query result  $Q(D)$  is exactly the Cartesian product of  $\mathcal{U} \times \mathcal{U}$ .

Consider a sub-database  $D'$  of  $D$  constructed above. Let  $R'_j$  be the corresponding sub-relation of  $R_j$  in  $D'$ . A solution  $D'$  to  $\text{SWP}(Q, D)$  is integral if  $R'_j = (\pi_{\text{attr}(R_j) \cap P'} R'_j) \times (\pi_{A_k} R'_j)$  holds for every relation  $R_j$  with  $A_k \in \text{attr}(R_j)$  and  $\text{attr}(R_j) \cap P' \neq \emptyset$ . Applying a similar argument as Lemma 21, we can show that there always exists an integral solution to  $\text{SWP}(Q, D)$ . Below, it suffices to focus on integral solutions. It can be easily proved that  $(\mathcal{U}, \mathcal{S})$  has a cover of size  $\leq k$  if and only if  $\text{SWP}(Q, D)$  has an integral solution of size  $\leq nq_1 + knq_2 + q_3$  where  $q_1, q_2, q_3 \leq |\text{rels}(Q)|$  are query-dependent parameters. If  $\text{SWP}$  is  $(1 - o(1)) \cdot \log N$ -approximatable for  $Q$ , there is a poly-time algorithm that can approximate set cover instances of input size  $n$  within a  $\log n$ -factor, which is impossible unless  $\text{P} = \text{NP}$ . ◀

**Connection with Non-Free-connex CQs.** We point out that every acyclic but non-free-connex CQ has a free sequence [5], hence does not have head-domination property. Together with Lemma 19, our characterization of  $\text{SWP}$  for acyclic CQs coincides with the separation between free-connex and non-free-connex CQs. In short,  $\text{SWP}$  is poly-time solvable or  $O(1)$ -approximatable for free-connex CQs, while no poly-time algorithm can approximate  $\text{SWP}$  for any acyclic but non-free-connex CQs within a factor of  $(1 - o(1)) \cdot \log N$ , unless  $\text{P} = \text{NP}$ .

## 4.2.2 Nested Clique

Although free sequence suffices to capture the hardness of approximating  $\text{SWP}$  for acyclic CQs, it is not enough for cyclic CQs. Let's start with the simplest cyclic CQ  $Q_{\text{pyramid}}(A, B, C) : -R_1(A, B), R_2(A, C), R_3(B, C), R_4(A, F), R_5(B, F), R_6(C, F)$  that does not contain a free sequence, but  $\text{SWP}$  is still difficult to approximate.

► **Lemma 25.** *There is no poly-time algorithm to approximate  $\text{SWP}$  for  $Q_{\text{pyramid}}$  within a factor of  $(1 - o(1)) \cdot \log N$ , unless  $\text{P} = \text{NP}$ .*

**Proof.** Consider an instance  $(\mathcal{U}, \mathcal{S})$  of set cover, with  $\mathcal{U} = \{u_1, u_2, \dots, u_n\}$  and  $\mathcal{S} = \{S_1, S_2, \dots, S_m\}$ , where  $m = n^c$  for some constant  $c > 1$ . We construct a database  $D$  for  $Q$  as follows. Let  $\text{dom}(A) = \{a_1, a_2, \dots, a_n\}$ ,  $\text{dom}(B) = \{b_1, b_2, \dots, b_n\}$ ,  $\text{dom}(C) = \{c_1, c_2, \dots, c_n\}$  and  $\text{dom}(F) = \text{dom}(F^-) \times \text{dom}(F^+)$ , where  $\text{dom}(F^-) = \{f_1^-, f_2^-, \dots, f_m^-\}$  and  $\text{dom}(F^+) = \{f_1^+, f_2^+, \dots, f_n^+\}$ . Relations  $R_1, R_2, R_3$  and  $R_6$  are Cartesian products of their corresponding attributes. For each pair  $(u_\ell, S_j) \in \mathcal{U} \times \mathcal{S}$  with  $u_\ell \in S_j$ , we add tuples of  $\{(a_\ell, f_j^-)\} \times \text{dom}(F^+)$  to  $R_4$ . For each  $i \in [n]$ , we add tuples  $\{b_i\} \times \text{dom}(F^-) \times \{f_i^+\}$  to  $R_5$ . It can be easily checked that the input size of  $D$  is  $O(n^{2c})$ , hence  $\log N = \Theta(\log n)$ .  $Q(D)$  is the Cartesian product between  $A, B$  and  $C$ . Hence, every solution to  $\text{SWP}(Q, D)$  includes all tuples in  $R_1, R_2, R_3$ . Below, we focus on  $R_4, R_5, R_6$ .

We observe that  $D$  enjoys highly symmetric structure over  $\text{dom}(B)$ . More specifically, each value  $b_i \in \text{dom}(B)$  induces a subquery  $Q_i(A, C) = R_4(A, F_i) \bowtie R_5(F_i) \bowtie R_6(C, F_i)$ , where  $\text{dom}(F_i) = \text{dom}(F^-) \times \{f_i^+\}$ , and a sub-database  $D^i = \{R_4^i, R_5^i, R_6^i\}$ , where  $R_4^i = \{(a_\ell, f_j^-, f_i^+) : \forall \ell \in [n], j \in [m], u_\ell \in S_j\}$ ,  $R_5^i = \text{dom}(F_i)$  and  $R_6^i = \text{dom}(C) \times \text{dom}(F_i)$ . It can be easily checked that  $\text{SWP}(Q, D) = R_1 \uplus R_2 \uplus R_3 \uplus (\uplus_{i \in [n]} \text{SWP}(Q_i, D^i))$ . For any  $i \in [n]$ , computing  $\text{SWP}(Q_i, D^i)$  is almost the same as  $Q_{\text{matrix}}$ . Moreover, the solution to each  $\text{SWP}(Q_i, D^i)$  shares the same structure, which is independent of the specific value  $b_i \in \text{dom}(B)$ . In a sub-database  $D' \subseteq D$ , let  $R'_i$  be the corresponding sub-relation of  $R_i$ . A solution  $D'$  to  $\text{SWP}(Q, D)$  is *integral* if  $R'_4 = (\pi_{A, F^-} R'_4) \times \text{dom}(F^+)$ ,  $R'_5 = \text{dom}(B) \times (\pi_{F^-} R'_5) \times \text{dom}(F^+)$ , and  $R'_6 = \text{dom}(C) \times (\pi_{F^-} R'_6) \times \text{dom}(F^+)$ . Implied by Lemma 21 and analysis above, there always exists an integral solution to  $\text{SWP}(Q, D)$ .

## 24:14 Finding Smallest Witnesses for Conjunctive Queries

Moreover,  $(\mathcal{U}, \mathcal{S})$  has a cover of size  $\leq k$  if and only if  $\text{SWP}(Q, D)$  has an integral witness of size  $\leq 3n^2 + n(n + k + kn) = (k + 4)n^2 + kn$ . If  $\text{SWP}$  is  $(1 - o(1)) \cdot \log N$ -approximable for  $Q$ , then there is a poly-time algorithm that can approximate set cover instances of input size  $n$  within a factor of  $(1 - o(1)) \cdot \log n$ , which is impossible unless  $\text{P} = \text{NP}$ . ◀

Now, we are ready to introduce the structure of *nested clique* and the *rename* procedure for capturing the hardness of cyclic CQs:

► **Definition 26** (Nested Clique). *In a CQ  $Q$ , a nested clique is a subset of attributes  $P \subseteq \text{attr}(Q)$  such that*

- *for any pair of attributes  $A, B \in P$ , there is some  $R_j \in \text{rels}(Q)$  with  $A, B \in \text{attr}(R_j)$ ;*
- *$P \cap \text{head}(Q) \neq \emptyset$  and  $P - \text{head}(Q) \neq \emptyset$ ;*
- *there is no relation  $R_j \in \text{rels}(Q)$  with  $P \cap \text{head}(Q) \subseteq \text{head}(R_j)$ .*

► **Definition 27** (Rename). *Given the nonout-connectivity graph  $H_Q$  of a CQ  $Q$  with connected components  $H_1, H_2, \dots, H_k$ , the rename procedure assigns one distinct attribute to all attributes in the same component. The resulted CQ  $Q'$  contains the same output attributes as  $Q$ , and each  $R_i \in \text{rels}(Q)$  defines a new relation  $R'_i \in \text{rels}(Q')$  with  $\text{attr}(R'_i) = \text{head}(R_i) \cup \{F_j : \forall j \in [k], H_j \cap \text{attr}(R_i) \neq \emptyset\}$ .*

In Figure 3,  $Q_1$  is renamed as  $Q'_1(A_1, A_2, A_3) : -R_1(A_1, F_1), R_2(F_1), R_3(A_2, F_1), R_4(A_2, A_3, F_2), R_5(A_1, A_2)$ .

► **Theorem 28.** *For a CQ  $Q$ , if its renamed query  $Q'$  contains a nested clique, there is no poly-time algorithm that can approximate  $\text{SWP}$  for  $Q$  within a factor of  $(1 - o(1)) \cdot \log N$ , unless  $\text{P} = \text{NP}$ .*

**Proof.** Let  $P \subseteq \text{attr}(Q)$  be the subset of attributes corresponding to the clique in the renamed query of  $Q$ . We identify the relation that contains the most number of output attributes of  $P$ , say  $R_2 = \arg \max_{R_i \in \text{rels}(Q)} |\text{attr}(R_i) \cap P \cap \text{head}(Q)|$ . As there exists no relation  $R_k \in \text{rels}(Q)$  with  $\text{head}(Q) \cap P \subseteq \text{attr}(R_k)$ , it is always feasible to identify an attribute  $A \in \text{head}(Q) \cap P - \text{attr}(R_2)$ . For simplicity, we denote  $P' = \text{head}(Q) \cap P - \text{attr}(R_2) - \{A\} = \{A_1, A_2, \dots, A_\ell\}$ , for some integer  $\ell$ . All non-output attributes in  $P - \text{head}(Q)$  collapse to a single attribute  $F$ . All other attributes in  $\text{attr}(Q) - P$  contain a dummy value  $\{*\}$ .

Consider an arbitrary instance of set cover  $(\mathcal{U}, \mathcal{S})$  with  $\mathcal{U} = \{u_1, u_2, \dots, u_n\}$  and  $\mathcal{S} = \{S_1, S_2, \dots, S_m\}$ , where  $m = n^c$  for some constant  $c > 1$ . We construct a database  $D$  for  $Q$ . Let  $\text{dom}(A) = \{a_1, a_2, \dots, a_n\}$ ,  $\text{dom}(B) = \{b_1, b_2, \dots, b_n\}$ ,  $\text{dom}(C) = \{c_1, c_2, \dots, c_n\}$  for every  $C \in P'$ , and  $\text{dom}(F) = \text{dom}(F^-) \times \text{dom}(F^+)$  where  $\text{dom}(F^-) = \{f_1^-, f_2^-, \dots, f_m^-\}$  and  $\text{dom}(F^+) = \{f_{h_1, h_2, \dots, h_\ell}^+ : \forall h_1, h_2, \dots, h_\ell \in [n]\}$ . We distinguish the following cases:

- If  $\text{attr}(R_i) \cap P \subseteq \text{head}(Q)$ ,  $R_i$  is a Cartesian product over all attributes in  $\text{head}(R_i) \cap P$ ;
- Otherwise, we further distinguish the following three cases:
  - $R_2$  is a Cartesian product over all attributes in  $\text{attr}(R_2) \cap P$ ;
  - If  $A, F \in \text{attr}(R_i)$ , we construct sub-relation  $(\pi_{A, F} R_i)$  such that for each pair  $(u_\ell, S_j) \in \mathcal{U} \times \mathcal{S}$  with  $u_\ell \in S_j$ , we add tuples  $\{a_\ell, f_j^-\} \times \text{dom}(F^+)$  to  $(\pi_{A, F} R_i)$ . If  $\text{attr}(R_i) \cap P' \neq \emptyset$ , for each tuple  $(a_\ell, f_{h_1, h_2, \dots, h_\ell}^+, f_j^-) \in (\pi_{A, F} R_i)$ , we extend it by attaching value  $c_{h_j}$  for attribute  $A_j \in \text{attr}(R_i) \cap P'$ . This way, we already obtain  $(\pi_{A, F, \text{attr}(R_i) \cap P'} R_i)$ . At last, we construct  $R_i$  as the Cartesian product of remaining attributes in  $\text{head}(R_i) \cap \text{head}(R_2) \cap P$  and  $(\pi_{A, F, \text{attr}(R_i) \cap P'} R_i)$ .

- Otherwise,  $F \in \text{attr}(R_i)$  but  $A \notin \text{attr}(R_i)$ . We construct sub-relation  $(\pi_F R_i) = \text{dom}(F)$ . If  $\text{attr}(R_i) \cap P' \neq \emptyset$ , for each tuple  $(f_{h_1}^+, f_{h_2}^+, \dots, f_{h_\ell}^+, f_j^-) \in \pi_F R_i$ , we extend it by attaching value  $c_{h_j}$  for attribute  $A_j \in \text{attr}(R_i) \cap P'$ . This way, we already obtain  $(\pi_{F, \text{attr}(R_i) \cap P'} R_i)$ . At last, we construct  $R_i$  as the Cartesian product of remaining attributes in  $\text{head}(R_i) \cap \text{head}(R_2) \cap P$  and  $(\pi_{A, F, \text{attr}(R_i) \cap P'} R_i)$ .

It can be checked that every relation contains  $O(n^{|\text{head}(Q) \cap P|} \cdot m)$  tuples, hence  $\log N = \Theta(\log n)$ . Meanwhile, the query result  $Q(D)$  is the Cartesian product over attributes in  $\text{head}(Q) \cap P$ , so every solution to  $\text{SWP}(Q, D)$  must contain all tuples in  $R_i$  if  $\text{attr}(R_i) \cap P \subseteq \text{head}(Q)$ , and the dummy tuple  $\{*\}$  in every relation  $R_i$  if  $\text{attr}(R_i) \cap P = \emptyset$ .

Here,  $D$  also enjoys highly symmetric structure over every attribute  $C \in P'$ . More specifically, every tuple  $t = (c_{i_1}, c_{i_2}, \dots, c_{i_\ell}) \in \times_{C \in P'} \text{dom}(C)$  induces a subquery by removing all attributes in  $P'$ , and restricting  $\text{dom}(F)$  as  $\text{dom}(F_t) = \text{dom}(F^-) \times \{f_{i_1}^+, f_{i_2}^+, \dots, f_{i_\ell}^+\}$ . In a sub-database  $D' \subseteq D$ , let  $R'_i$  be the corresponding sub-relation  $R_i$ . A solution  $D'$  to  $\text{SWP}(Q, D)$  is *integral* if:

- for every relation  $R_i \in \text{rels}(Q)$  with  $A, F \in \text{attr}(R_i)$ ,

$$\pi_{A, F, \text{head}(R_i) \cap \text{head}(R_2) \cap P} R'_i = (\pi_{A, F^-} R'_i) \times \text{dom}(F^+) \times (\times_{C \in \text{head}(R_i) \cap \text{head}(R_2) \cap P} \text{dom}(C))$$

- for every relation  $R_i \in \text{rels}(Q)$  with  $A \notin \text{attr}(R_i)$  and  $F \in \text{attr}(R_i)$ ,

$$\pi_{F, \text{head}(R_i) \cap \text{head}(R_2) \cap P} R'_i = (\pi_{F^-} R'_i) \times \text{dom}(F^+) \times (\times_{C \in \text{head}(R_i) \cap \text{head}(R_2) \cap P} \text{dom}(C))$$

It can be easily shown that there always exists an integral solution to  $\text{SWP}(Q, D)$ . Moreover,  $(\mathcal{U}, \mathcal{S})$  has a cover of size  $\leq k$  if and only if  $\text{SWP}(Q, D)$  has an integral solution of size  $(q_1 k + q_2) \cdot n^{|\text{head}(Q) \cap P| - 1}$ , where  $q_1, q_2 \leq |\text{rels}(Q)|$  are some query-dependent parameters. Applying a similar argument as Lemma 21, we can show that if  $\text{SWP}$  is  $(1 - o(1)) \cdot \log N$ -approximable for  $Q$ , there is a poly-time algorithm that can approximate set cover instances of input size  $n$  within a  $\log n$ -factor, which is impossible unless  $\text{P} = \text{NP}$ . ◀

### 4.3 Completeness

At last, we complete the proof of Theorem 16 by establishing the connection between the non-existence of hardcore structures and head-domination property:

► **Lemma 29.** *In a CQ  $Q$ , if there is neither a free sequence nor a nested clique in its renamed query,  $Q$  has head-domination property.*

We have proved Lemma 29 for acyclic CQs, such that if  $Q$  does not contain a free sequence,  $Q$  has head-cluster property. It suffices to focus on cyclic CQs. We note that any cyclic CQ contains a *cycle* or *non-conformal clique* [8]. Our proof is based on a technical lemma that if every cycle or non-conformal clique contains only output attributes or only non-output attributes,  $Q$  has head-domination property. We complete it by showing that if  $Q$  does not contain a free sequence or nested clique in its renamed query, no cycle or non-conformal clique contains both output and non-output attributes.

## 5 Approximation Algorithms

We next explore possible approximation algorithms for CQs without head-domination property. All missing proofs are in the full version [28].

## 5.1 Baseline

A baseline for general CQs returns the union of smallest witness for every query result, which includes at most  $\min\{N, |\text{rels}(Q)| \cdot |Q(D)|\}$  tuples. Meanwhile, AGM bound [4] implies that at least  $O(|Q(D)|^{1/\rho^*})$  tuples are needed to reproduce  $|Q(D)|$  results, where  $\rho^*$  is the fractional edge covering number of  $Q$ . Together, we obtain:

► **Theorem 30.** *SWP is  $N^{1-1/\rho^*}$ -approximable for any CQ  $Q$ , where  $\rho^*$  is the fractional edge covering number of  $Q$ .*

This upper bound is polynomially larger than the logarithmic lower bound proved in Section 4. We next explore better approximation algorithms for some commonly-used CQs.

## 5.2 Star CQs

We look into one commonly-used class of CQs noted as *star* CQs:

$$Q_{\text{star}}(A_1, A_2, \dots, A_m) : -R_1(A_1, B), R_2(A_2, B), \dots, R_m(A_m, B).$$

Our approximation algorithm follows the greedy strategy developed for weighted set cover problem, where each element to be covered is a query result and each subset is a collection of tuples. Intuitively, the greedy strategy always picks a subset that minimizes the “price” for covering the remaining uncovered elements. The question boils down to specifying the universe  $\mathcal{U}$  of elements, and the family  $\mathcal{S}$  of subsets as well as their weights. However, naively taking every possible sub-collection of tuples from the input database as a subset, would generate an exponentially large  $\mathcal{S}$ , which leads to a greedy algorithm running in exponential time. Hence, it is critical to keep the size of  $\mathcal{S}$  small. Let’s start with  $Q_{\text{matrix}}$  ( $m = 2$ ).

**Greedy Algorithm for  $Q_{\text{matrix}}$ .** Given  $Q_{\text{matrix}}$ , a database  $D$ , and a subset of query results  $\mathcal{C} \subseteq Q_{\text{matrix}}(D)$ , the price of a collections of tuples  $(X, Y)$  for  $X \subseteq R_1$  and  $Y \subseteq R_2$  is defined as  $f(\mathcal{C}, X, Y) = \frac{|X|+|Y|}{|\pi_{A,C}(X \bowtie Y) - \mathcal{C}|}$ . To shrink the space of candidate subsets, a critical observation is that the subset with minimum price chosen by the greedy algorithm cannot be *divided* further, i.e., all tuples should have the same join value as captured by Lemma 31.

► **Lemma 31.** *Given  $Q_{\text{matrix}}$  and a database  $D$ , for two distinct values  $b, b' \in \text{dom}(B)$ , and two pairs of subsets of tuples  $(X_1, Y_1) \in 2^{\sigma_{B=b}R_1} \times 2^{\sigma_{B=b}R_2}$ ,  $(X_2, Y_2) \in 2^{\sigma_{B=b'}R_1} \times 2^{\sigma_{B=b'}R_2}$ , for arbitrary  $\mathcal{C} \subseteq Q_{\text{matrix}}(D)$ ,  $\min\{f(\mathcal{C}, X_1, Y_1), f(\mathcal{C}, X_2, Y_2)\} \leq f(\mathcal{C}, X_1 \cup X_2, Y_1 \cup Y_2)$ .*

Now, we are ready to present a greedy algorithm that runs in polynomial time. As shown in Algorithm 2, we always maintain a pair  $(X_i, Y_i)$  for every value  $b_i \in \text{dom}(B)$ , such that  $(X_i, Y_i)$  minimize the function  $f(\mathcal{C}, X, Y)$  for  $X \subseteq \sigma_{B=b_i}R_1$  and  $Y \subseteq \sigma_{B=b_i}R_2$ . The greedy algorithm always chooses the one pair with minimum price, pick all related tuples in this pair, and update the coverage  $\mathcal{C}$ . After this step, we also need to update the candidate pair  $(X_i, Y_i)$  for each value  $b_i \in \text{dom}(B)$  and enter into the next iteration. We will stop until all query results are covered. The remaining question is how to compute  $(X_i, Y_i)$  with minimum price (line 8) efficiently. We mention *densest subgraph in the bipartite graph* in the literature [25, 29], for which a poly-time algorithm based on max-flow has been proposed.

► **Definition 32 (Densest Subgraph in Bipartite Graph).** *Given a bipartite graph  $(X, Y, E)$  with  $E : X \times Y \rightarrow \{0, 1\}$ , it asks to find  $X' \subseteq X, Y' \subseteq Y$  to maximize  $\frac{\sum_{x \in X', y \in Y'} E(x, y)}{|X'| + |Y'|}$ .*



---

**Algorithm 2** GREEDYSWP( $Q_{\text{matrix}}, D$ ).

---

```

1  $D' \leftarrow \emptyset, \mathcal{C} \leftarrow \emptyset;$ 
2 foreach  $b_i \in \text{dom}(B)$  do  $(X_i, Y_i) \leftarrow (\sigma_{B=b_i} R_1, \sigma_{B=b_i} R_2);$ 
3 while  $\mathcal{C} \neq Q_{\text{matrix}}(D)$  do
4    $b_j \leftarrow \arg \min_{b_i \in \text{dom}(B)} f(\mathcal{C}, X_i, Y_i);$ 
5    $D' \leftarrow D' \cup X_j \cup Y_j, \mathcal{C} \leftarrow \mathcal{C} \cup \pi_{A, \mathcal{C}}(X_j \bowtie Y_j);$ 
6   foreach  $b_i \in \text{dom}(B)$  do
7      $(X_i, Y_i) \leftarrow \arg \min_{X \subseteq \sigma_{B=b_i} R_1, Y \subseteq \sigma_{B=b_i} R_2} f(\mathcal{C}, X, Y);$ 
8 return  $D';$ 

```

---

Back to our problem, each value  $b_i \in \text{dom}(B)$  induces a bipartite graph with  $X = \sigma_{B=b_i} R_1$ ,  $Y = \sigma_{B=b_i} R_2$ , and  $E = \{(x, y) \mid x \in X, y \in Y, \pi_{A, \mathcal{C}}(x \bowtie y) \notin \mathcal{C}\}$ . This way, the densest subgraph in this bipartite graph corresponds to a subset of uncovered query results with minimum price, and vice versa.

The approximation ratio of our greedy algorithm follows the standard analysis of weighted set cover [39]. We next focus on the time complexity. The while-loop proceeds in at most  $O(|Q(D)|)$  iterations, since  $|\mathcal{C}|$  increases by at least 1 in every iteration. It takes  $O(N)$  time to find the pair with minimum price, since there are  $O(N)$  distinct values in  $\text{dom}(B)$ . Moreover, it takes polynomial time to update  $(X_i, Y_i)$  for each  $b_i \in \text{dom}(B)$ . Overall, Algorithm 2 runs in polynomial time in terms of  $N$ .

**Extensions.** Our algorithm for  $Q_{\text{matrix}}$  can be extended to star CQs, and further to all CQs with only one non-output attribute. Similar property as Lemma 31 also holds. Our greedy algorithm needs a generalized primitive, noted as *densest subgraph in hypergraph*<sup>5</sup> [27] for finding the “set” with the smallest price. Following the similar analysis, we obtain:

► **Theorem 33.** *For any CQ  $Q$  with  $|\text{attr}(Q) - \text{head}(Q)| = 1$ , SWP is  $O(\log N)$ -approximable.*

### 5.3 Line CQs

We turn to another commonly-used class of CQs noted as *line* CQs:

$$Q_{\text{line}}(A_1, A_{m+1}) : -R_1(A_1, A_2), R_2(A_2, A_3), \dots, R_m(A_m, A_{m+1})$$

with  $m \geq 3$ . We surprisingly find that SWP for line CQs is closely related to *directed Steiner forest* (DSF) problem [18, 11, 21, 16] in the network design: Given an edge-weighted directed graph  $G = (V, E)$  of  $|V| = n$  and a set of  $k$  demand pairs  $\{(s_i, t_i) \in V \times V : i \in [k]\}$  and the goal is to find a subgraph  $G'$  of  $G$  with minimum weight such that there is a path in  $G'$  from  $s_i$  to  $t_i$  for every  $i \in [k]$ . Observe that  $\text{SWP}(Q_{\text{line}}, D)$  is a special case of DSF. This way, all existing algorithm proposed for DSF can be applied to SWP for  $Q_{\text{line}}$ . The best approximation ratio achieved is  $O(\min\{k^{\frac{1}{2}+o(1)}, n^{0.5778}\})$  [21, 11, 1]. Combining the baseline in Section 5.1 (with  $\rho^* = 2$  for line CQs) and existing algorithms for DSF, we obtain:

► **Theorem 34.** *For  $Q_{\text{line}}$  and any database  $D$  of input size  $N$ , there is a poly-time algorithm that can approximate  $\text{SWP}(Q_{\text{line}}, D)$  within a factor of  $O(\min\{|Q_{\text{line}}(D)|^{\frac{1}{2}+o(1)}, \text{dom}^{0.5778}, N^{\frac{1}{2}}\})$ , where  $\text{dom}$  is the number of values that participates in at least one full join result.*

---

<sup>5</sup> Given a hypergraph  $H = (V, E)$  for  $E \subseteq 2^V$ , it asks to find a subset of nodes  $S \subseteq V$  such that the ratio  $|\{e \in E : e \subseteq S\}|/|S|$  is maximized.



There is a large body of works investigating the lower bounds of DSF; and we refer interested readers to [16] for details. We mention a polynomial lower bound  $\Omega(k^{1/4-o(1)})$  for DSF, but it cannot be applied to SWP, since SWP is a special case with its complexity measured by the number of edges in the graph. Instead, we built a reduction from *label cover* to SWP for  $Q_{\text{line}}$  directly (which is an adaption of reduction proposed in [18, 13]) and prove the following:

► **Theorem 35.** *No poly-time algorithm approximates SWP for  $Q_{\text{line}}$  within  $\Omega(2^{(\log N)^{1-\epsilon}})$  factor for any constant  $\epsilon > 0$ , unless  $P = NP$ .*

## 6 Related Work

**Factorized database [34].** Factorized database studies a nested representations of query results that can be exponentially more succinct than flat query result, which has the same goal as SWP. They built a tree-based representations by exploiting the distributivity of product over union and commutativity of product and union. This notion is quite different from SWP. They measure the regularity of factorizations by readability, the minimum over all its representations of the maximum number of occurrences of any tuple in that representation, while SWP measures the size of witness.

**Data Synopses [14, 36].** In approximate query processing, people studied a lossy, compact synopsis of the data such that queries can be efficiently and approximately executed against the synopsis rather than the entire dataset, such as random samples, sketches, histograms and wavelets. These data synopses differ in terms of what class of queries can be approximately answered, space usage, accuracy etc. In computational geometry, a corset is a small set of points that approximate the shape of a larger point set, such as for shape-fitting, density estimation, high-dimensional vectors or points, clustering, graphs, Fourier transforms etc. SWP can also be viewed as data synopsis, since it also selects a representative subset of tuples. But, it is more query-dependent, as different CQs over the same database (such as  $Q_1(x) : -R_1(x, y), R_2(y, z)$  and  $Q_2(x, y, z) : -R_1(x, y), R_2(y, z)$ ) can have dramatically different SWP. Furthermore, all query results must be preserved by SWP, but this is never guaranteed in other data synopses. There is no space-accuracy tradeoff in SWP as well.

**Related to Other Problems in Database Theory.** The SWP problem also outputs the smallest provenance that can explain why all queries results are correct. This notion of why-provenance has been extensively investigated in the literature [9, 26, 3], but not from the perspective of minimizing the size of witness. The SWP problem is also related to the resilience problem [22, 23, 37], which intuitively finds the smallest number of tuples to remove so that the query answer turns into false. Our SWP problem essentially finds the maximum number of tuples to remove while the query answer does not change. We can observe clear connection here, but their solutions do not imply anything to each other.

## 7 Conclusion

In this paper, we study the data complexity of SWP problem for CQs without self-joins. There are several interesting problems left:

1. *Approximating SWP for CQs without head-domination property.* So far, the approximation of SWP is well-understood only on some specific class of CQs without head-domination property. For remaining CQs, both upper and lower bounds remain to be improved, which may lead to fundamental breakthrough for other related problems, such as DSF.

2. *SWP for CQs with self-joins*. It becomes much more challenging when self-joins exists, as one tuple appears in multiple logical copies of input relation. Similar observation has been made for the related resilience problem [23, 31]
3. *Relaxing the number of query results witnessed*. It is possible to explore approximation on the number of query results that can be witnessed. Here, SWP is related to the *partial set cover* problem, for which many approximation algorithms [24] have been studied.

---

## References

- 1 Amir Abboud and Greg Bodwin. Reachability preservers: New extremal bounds and approximation algorithms. In *SODA*, pages 1865–1883. SIAM, 2018. doi:10.1137/1.9781611975031.122.
- 2 Mahmoud Abo Khamis, Hung Q Ngo, and Atri Rudra. Faq: questions asked frequently. In *PODS*, pages 13–28, 2016. doi:10.1145/2902251.2902280.
- 3 Yael Amsterdamer, Daniel Deutch, and Val Tannen. Provenance for aggregate queries. In *PODS*, pages 153–164, 2011. doi:10.1145/1989284.1989302.
- 4 Albert Atserias, Martin Grohe, and Dániel Marx. Size bounds and query plans for relational joins. In *FOCS*, FOCS '08, pages 739–748, 2008. doi:10.1109/FOCS.2008.43.
- 5 Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. On acyclic conjunctive queries and constant delay enumeration. In *CSL*, pages 208–222. Springer, 2007. doi:10.1007/978-3-540-74915-8\_18.
- 6 C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *JACM*, 30(3):479–513, 1983. doi:10.1145/2402.322389.
- 7 KORTE Bernhard and JENS Vygen. Combinatorial optimization: Theory and algorithms. *Springer, Third Edition, 2005.*, 2008.
- 8 Johann Brault-Baron. Hypergraph acyclicity revisited. *CSUR*, 49(3):1–26, 2016. doi:10.1145/2983573.
- 9 Peter Buneman, Sanjeev Khanna, and Tan Wang-Chiew. Why and where: A characterization of data provenance. In *ICDT*, pages 316–330. Springer, 2001. doi:10.1007/3-540-44503-X\_20.
- 10 Surajit Chaudhuri, Rajeev Motwani, and Vivek Narasayya. On random sampling over joins. *ACM SIGMOD Record*, 28(2):263–274, 1999. doi:10.1145/304182.304206.
- 11 Chandra Chekuri, Guy Even, Anupam Gupta, and Danny Segev. Set connectivity problems in undirected graphs and the directed steiner network problem. *TALG*, 7(2):1–17, 2011. doi:10.1145/1921659.1921664.
- 12 Yu Chen and Ke Yi. Random sampling and size estimation over cyclic joins. In *ICDT*, 2020. doi:10.4230/LIPIcs.ICDT.2020.7.
- 13 Rajesh Chitnis, Andreas Emil Feldmann, and Pasin Manurangsi. Parameterized approximation algorithms for bidirected steiner network problems. *ACM Trans. Algorithms*, 17(2), apr 2021. doi:10.1145/3447584.
- 14 Graham Cormode, Minos Garofalakis, Peter J Haas, Chris Jermaine, et al. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends® in Databases*, 4(1–3):1–294, 2011. doi:10.1561/1900000004.
- 15 Graham Cormode and Ke Yi. *Small summaries for big data*. Cambridge University Press, 2020.
- 16 Irit Dinur and Pasin Manurangsi. Eth-hardness of approximating 2-csp and directed steiner network. In Anna R. Karlin, editor, *ITCS*, volume 94 of *LIPIcs*, pages 36:1–36:20, 2018. doi:10.4230/LIPIcs.ITCS.2018.36.
- 17 Irit Dinur and David Steurer. Analytical approach to parallel repetition. In *STOC*, pages 624–633, 2014. doi:10.1145/2591796.2591884.
- 18 Yevgeniy Dodis and Sanjeev Khanna. Design networks with bounded pairwise distance. In *STOC*, pages 750–759, 1999. doi:10.1145/301250.301447.
- 19 R. Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *JACM*, 30(3):514–550, 1983. doi:10.1145/2402.322390.

- 20 Uriel Feige. A threshold of  $\ln n$  for approximating set cover. *JACM*, 45(4):634–652, 1998.
- 21 Moran Feldman, Guy Kortsarz, and Zeev Nutov. Improved approximation algorithms for directed steiner forest. *JCSS*, 78(1):279–292, 2012. doi:10.1016/j.jcss.2011.05.009.
- 22 Cibele Freire, Wolfgang Gatterbauer, Neil Immerman, and Alexandra Meliou. The complexity of resilience and responsibility for self-join-free conjunctive queries. *PVLDB*, 9(3):180–191, 2015. doi:10.14778/2850583.2850592.
- 23 Cibele Freire, Wolfgang Gatterbauer, Neil Immerman, and Alexandra Meliou. New results for the complexity of resilience for binary conjunctive queries with self-joins. In *PODS*, pages 271–284, 2020. doi:10.1145/3375395.3387647.
- 24 Rajiv Gandhi, Samir Khuller, and Aravind Srinivasan. Approximation algorithms for partial covering problems. *Journal of Algorithms*, 53(1):55–84, 2004. doi:10.1016/j.jalgor.2004.04.002.
- 25 Andrew V Goldberg. Finding a maximum density subgraph. Technical report, University of California Berkeley, 1984.
- 26 Todd J Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. In *PODS*, pages 31–40, 2007. doi:10.1145/1265530.1265535.
- 27 Shuguang Hu, Xiaowei Wu, and TH Hubert Chan. Maintaining densest subsets efficiently in evolving hypergraphs. In *CIKM*, pages 929–938, 2017. doi:10.1145/3132847.3132907.
- 28 Xiao Hu and Stavros Sintos. Finding smallest witnesses for conjunctive queries. *arXiv preprint*, 2023. doi:10.48550/arXiv.2311.18157.
- 29 Samir Khuller and Barna Saha. On finding dense subgraphs. In *ICALP*, pages 597–608. Springer, 2009. doi:10.1007/978-3-642-02927-1\_50.
- 30 Benny Kimelfeld, Jan Vondrák, and Ryan Williams. Maximizing conjunctive views in deletion propagation. In *PODS*, pages 187–198, 2011. doi:10.1145/1989284.1989308.
- 31 Benny Kimelfeld, Jan Vondrák, and Ryan Williams. Maximizing conjunctive views in deletion propagation. *TODS*, 37(4):1–37, 2012. doi:10.1145/2389241.2389243.
- 32 Zhengjie Miao, Sudeepa Roy, and Jun Yang. Explaining wrong queries using small examples. In *SIGMOD*, pages 503–520, 2019. doi:10.1145/3299869.3319866.
- 33 Dana Moshkovitz. The projection games conjecture and the np-hardness of  $\ln n$ -approximating set-cover. In *APPROX-RANDOM*, pages 276–287. Springer, 2012. doi:10.1007/978-3-642-32512-0\_24.
- 34 Dan Olteanu and Maximilian Schleich. Factorized databases. *ACM SIGMOD Record*, 45(2):5–16, 2016. doi:10.1145/3003665.3003667.
- 35 John Paparrizos, Chunwei Liu, Bruno Barbarioli, Johnny Hwang, Ikraduya Edian, Aaron J Elmore, Michael J Franklin, and Sanjay Krishnan. Vergedb: A database for iot analytics on edge devices. In *CIDR*, 2021. URL: [http://cidrdb.org/cidr2021/papers/cidr2021\\_paper11.pdf](http://cidrdb.org/cidr2021/papers/cidr2021_paper11.pdf).
- 36 Jeff M Phillips. Coresets and sketches. In *Handbook of discrete and computational geometry*, pages 1269–1288. Chapman and Hall/CRC, 2017.
- 37 Biao Qin, Deying Li, and Chunlai Zhou. The resilience of conjunctive queries with inequalities. *Information Sciences*, 613:982–1002, 2022. doi:10.1016/j.ins.2022.08.049.
- 38 Moshe Y Vardi. The complexity of relational query languages. In *STOC*, pages 137–146, 1982. doi:10.1145/800070.802186.
- 39 Vijay V Vazirani. *Approximation algorithms*, volume 1. Springer, 2001.
- 40 Mihalis Yannakakis. Algorithms for acyclic database schemes. In *VLDB*, volume 81, pages 82–94, 1981.
- 41 Zhuoyue Zhao, Robert Christensen, Feifei Li, Xiao Hu, and Ke Yi. Random sampling over joins revisited. In *SIGMOD*, pages 1525–1539, 2018. doi:10.1145/3183713.3183739.

# Ranked Enumeration for MSO on Trees via Knowledge Compilation

**Antoine Amarilli** ✉ 🏠 

LTCI, Télécom Paris, Institut Polytechnique de Paris, France

**Pierre Bourhis** ✉ 

Univ. Lille, CNRS, Inria, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France

**Florent Capelli** ✉ 🏠 

Univ. Artois, CNRS, UMR 8188, Centre de Recherche en Informatique de Lens (CRIL), F-62300 Lens, France

**Mikaël Monet** ✉ 🏠 

Université de Lille, CNRS, Inria, UMR 9189 – CRIStAL, F-59000 Lille, France

---

## Abstract

We study the problem of enumerating the satisfying assignments for certain circuit classes from knowledge compilation, where assignments are ranked in a specific order. In particular, we show how this problem can be used to efficiently perform ranked enumeration of the answers to MSO queries over trees, with the order being given by a ranking function satisfying a subset-monotonicity property.

Assuming that the number of variables is constant, we show that we can enumerate the satisfying assignments in ranked order for so-called *multivalued circuits* that are smooth, decomposable, and in negation normal form (smooth multivalued DNNF). There is no preprocessing and the enumeration delay is linear in the size of the circuit times the number of values, plus a logarithmic term in the number of assignments produced so far. If we further assume that the circuit is deterministic (smooth multivalued d-DNNF), we can achieve linear-time preprocessing in the circuit, and the delay only features the logarithmic term.

**2012 ACM Subject Classification** Information systems → Relational database model

**Keywords and phrases** Enumeration, knowledge compilation, monadic second-order logic

**Digital Object Identifier** 10.4230/LIPIcs.ICDT.2024.25

**Related Version** *Full Version (with all Proofs)*: <https://arxiv.org/abs/2310.00731> [2]

**Funding** *Antoine Amarilli*: Partially supported by the ANR project EQUUS ANR-19-CE48-0019, by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 431183758, and by the ANR project ANR-18-CE23-0003-02 (“CQFD”). This work was done in part while the author was visiting the Simons Institute for the Theory of Computing.

*Florent Capelli*: This work was supported by project ANR KCODA, ANR-20-CE48-0004.

*Mikaël Monet*: This work was done in part while the author was visiting the Simons Institute for the Theory of Computing.

## 1 Introduction

Data management tasks often require the evaluation of queries on large datasets, in settings where the number of query answers may be very large. For this reason, the framework of *enumeration algorithms* has been proposed as a way to distinguish the *preprocessing time* of query evaluation algorithms and the maximal *delay* between two successive answers [32, 37]. Enumeration algorithms have been studied in several contexts: for conjunctive queries [8]



© Antoine Amarilli, Pierre Bourhis, Florent Capelli, and Mikaël Monet; licensed under Creative Commons License CC-BY 4.0

27th International Conference on Database Theory (ICDT 2024).

Editors: Graham Cormode and Michael Shekelyan; Article No. 25; pp. 25:1–25:18

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

and unions of conjunctive queries [10, 16] over relational databases; for first-order logic over bounded-degree structures [23], structures with local bounded expansion [33], and nowhere dense graphs [31]; and for monadic second-order logic (MSO) over trees [7, 25, 3].

We focus on the setting of MSO over trees. In this context, the following enumeration result is already known. For any fixed MSO query  $Q$  (i.e., in *data complexity*) where the free variables are assumed to be first-order, considering the answers of  $Q$  on a tree  $T$  given as input (i.e., the functions that map the variables of  $Q$  to nodes of  $T$  in a way that satisfies  $Q$ ), we can enumerate them with linear preprocessing on the tree  $T$  and with constant delay. If the free variables are second-order, then the delay is output-linear, i.e., linear in each produced answer [7, 3]. Further results are known when the query is not fixed but given as input as a potentially non-deterministic automaton [4, 5], or when maintaining the enumeration structure under tree updates [28, 5].

However, despite their favorable delay bounds, a shortcoming of these enumeration algorithms is that they enumerate answers in an opaque order which cannot be controlled. This is in contrast with application settings where answers should be enumerated, e.g., by decreasing order of relevance, or focusing on the *top-k* most relevant answers. This justifies the need for enumeration algorithms that can produce answers in a user-defined order, even if they do so at the expense of higher delay bounds.

This task, called *ranked enumeration*, has recently been studied in various contexts. For instance, Carmeli et al. [17, 13, 14] study for which order functions one can efficiently perform ranked direct access to the answers of conjunctive queries: here, efficient ranked direct access implies efficient ranked enumeration. Ranked enumeration has also been studied to support order-by operators on factorized databases [9]. Other works have studied ranked enumeration for document spanners [21], which relate to the evaluation of MSO queries over words. Closer to applications, some works have studied the ranked enumeration of conjunctive query answers, e.g., Deep et al. [20, 19] or Tziavelis et al. [35, 36]. Variants of in-order enumeration have been also studied on knowledge compilation circuit classes, for instance *top-k*, with a pseudo-polynomial time algorithm [11]. Closest to the present work, Bourhis et al. [12] have studied enumeration on *words* where the ranking function on answers is expressed in the formalism of MSO cost functions. They show that enumeration can be performed with linear preprocessing, with a delay between answers which is no longer constant but logarithmic in the size of the input word. However, their result does not apply in the more general context of trees.

**Contributions.** In this paper, we embark on the study of efficient ranked enumeration algorithms for the answers to MSO queries on trees, assuming that all free variables are first-order. We define this task by assigning *scores* to each so-called *singleton assignment* [ $x \rightarrow d$ ] describing that variable  $x$  is assigned tree node  $d$ , and combining these values into a *ranking function* while assuming a *subset-monotonicity property* [36]: intuitively, when extending two partial assignments in the same manner, then the order between them does not change. This setting covers many ranking functions, e.g., those defined by order, sum, or a lexicographic order on the variables. Our main contribution is then to show the following results on the data complexity of ranked enumeration for MSO queries on trees:

► **Result 1.** *For any fixed MSO query  $Q(x_1, \dots, x_n)$  with free first-order variables, given as input a tree  $T$  and a subset-monotone ranking function  $w$  on the partial assignments of  $x_1, \dots, x_n$  to nodes of  $T$ , we can enumerate the answers to  $Q$  on  $T$  in nonincreasing order of scores according to  $w$  with a preprocessing time of  $O(|T|)$  and a delay of  $O(\log(K + 1))$ , where  $K$  is the number of answers produced so far.*

Note that, as the total number of answers is at most  $|T|^n$ , and as  $n$  is constant in data complexity, the delay of  $O(\log(K+1))$  can alternatively be bounded by  $O(n \log |T|)$ , or  $O(\log |T|)$ . This matches the bound of [12] on words, though their notion of rank is different. Further, our bound shows that the first answers can be produced faster, e.g., for top- $k$  computation.

Our results for MSO queries on trees are shown in the general framework of *circuit-based enumeration methods*, introduced by [3]. In this framework, enumeration results are achieved by first translating the task to a class of structured circuits from knowledge compilation, and then proposing an enumeration algorithm that works directly on the structured class. This makes it possible to re-use enumeration algorithms across a variety of problems that compile to circuits. In this paper, as our task consists in enumerating assignments (from first-order variables of an MSO query to tree nodes), we phrase our results in terms of *multivalued circuits*. These circuits generalize Boolean circuits by allowing variables to take values in a larger domain than  $\{0, 1\}$ : intuitively, the domain will be the set of the tree nodes. We assume that circuits are *decomposable*, i.e., that no variable has a path to two different inputs of a  $\wedge$ -gate: this yields *multivalued DNNFs*, which generalize usual DNNFs. We also assume that the circuits are *smooth*: intuitively, no variable is omitted when combining partial assignments at an  $\vee$ -gate. Multivalued circuits can be smoothed while preserving decomposability, in quadratic time or faster in some cases [34]. Smooth multivalued DNNF circuits can alternatively be understood as factorized databases, but we do not impose that they are *normal* [30], i.e., the depth can be arbitrary.

Our enumeration task for MSO on trees thus amounts to the enumeration of satisfying assignments of smooth multivalued DNNFs, following a ranking function which we assume to be subset-monotone. However, we are not aware of existing results for ranked enumeration on circuits in the knowledge compilation literature. For this reason, the second contribution of this paper is to show efficient enumeration algorithms on these smooth multivalued DNNFs.

We first present an algorithm for this task that runs with no preprocessing and polynomial delay. The algorithm can be seen as an instance of the Lawler-Murty [26, 29] procedure. We show:

► **Result 2.** *For any constant  $n \in \mathbb{N}$ , given a smooth multivalued DNNF circuit  $C$  with domain  $D$  and with  $n$  variables, given a subset-monotone ranking function  $w$ , we can enumerate the satisfying assignments of  $C$  in nonincreasing order of scores according to  $w$  with delay  $O(|D| \times |C| + \log(K+1))$ , where  $K$  is the number of assignments produced so far.*

We then show a second algorithm, which allows for a better delay bound at the expense of making an additional assumption on the circuit; it is with this algorithm that we prove Result 1. The additional assumption is that the circuit is *deterministic*: intuitively, no partial assignment is captured twice. This corresponds to the class of *smooth multivalued d-DNNF circuits*. For our task of enumerating MSO query answers, the determinism property can intuitively be enforced on circuits when we compute them using an deterministic tree automaton to represent the query. We then show:

► **Result 3.** *For any constant  $n \in \mathbb{N}$ , given a smooth multivalued d-DNNF circuit  $C$  with  $n$  variables, given a subset-monotone ranking function  $w$ , we can enumerate the satisfying assignments of  $C$  in nonincreasing order of scores according to  $w$  with preprocessing time  $O(|C|)$  and delay  $O(\log(K+1))$ , where  $K$  is the number of assignments produced so far.*

**Paper structure.** We give preliminary definitions in Section 2. We first study in Section 3 the ranked enumeration problem for smooth multivalued DNNF circuits (Result 2). We then move on to a more efficient algorithm on smooth multivalued d-DNNF circuits (Result 3)



in Section 4. We show how to apply the second algorithm to ranked enumeration for the answers to MSO queries (Result 1) in Section 5. We conclude in Section 6. Missing proofs can be found in the full version [2].

## 2 Preliminaries

For  $n \in \mathbb{N}$ , we write  $[n]$  for the set  $\{1, \dots, n\}$ .

**Assignments.** For two finite sets  $D$  of *values* and  $X$  of *variables*, an *assignment on domain  $D$  and variables  $X$*  is a mapping from  $X$  to  $D$ . We write  $D^X$  the set of such assignments. We can see assignments as sets of *singleton assignments*, where a *singleton assignment* is an expression of the form  $[x \rightarrow d]$  with  $x \in X$  and  $d \in D$ .

Two assignments  $\tau \in D^Y$  and  $\sigma \in D^Z$  are *compatible*, written  $\tau \simeq \sigma$ , if we have  $\tau(x) = \sigma(x)$  for every  $x \in Y \cap Z$ . In this case, we denote by  $\tau \bowtie \sigma$  the assignment of  $D^{Y \cup Z}$  defined following the natural join, i.e., for  $y \in Y \setminus Z$  we set  $(\tau \bowtie \sigma)(y) := \tau(y)$ , for  $z \in Z \setminus Y$  we set  $(\tau \bowtie \sigma)(z) := \sigma(z)$ , and for  $x \in Z \cap Y$ , we set  $(\tau \bowtie \sigma)(x)$  to the common value  $\tau(x) = \sigma(x)$ . Two assignments  $\tau \in D^Y$  and  $\sigma \in D^Z$  are *disjoint* if  $Y \cap Z = \emptyset$ : then they are always compatible and  $\tau \bowtie \sigma$  corresponds to the relational product, which we write  $\tau \times \sigma$ .

Given  $R \subseteq D^Y$  and  $S \subseteq D^Z$ , we define  $R \wedge S = \{\tau \bowtie \sigma \mid \tau \in R, \sigma \in S, \tau \simeq \sigma\}$ : this is a subset of  $D^{Y \cup Z}$ . Note how, if the domain is  $D = \{0, 1\}$ , then this corresponds to the usual conjunction for Boolean functions, and in general we can see it as a relational join, or a relational product whenever  $Y \cap Z = \emptyset$ . Further, we define  $R \vee S = \{\tau \in D^{Y \cup Z} \mid \tau|_Y \in R \text{ or } \tau|_Z \in S\}$ , which is again a subset of  $D^{Y \cup Z}$ . Again observe how, when  $D = \{0, 1\}$ , this corresponds to disjunction; and in general we can see this as relational union except that assignments over  $Y$  and  $Z$  are each implicitly completed in all possible ways to assignments over  $Y \cup Z$ .

**Multivalued circuits.** A *multivalued circuit  $C$  on domain  $D$  and variables  $X$*  is a DAG with labeled vertices which are called *gates*. The circuit also has a distinguished gate  $r$  called the *output gate of  $C$* . Gates having no incoming edges are called *inputs* of  $C$ . Moreover, we have:

- Every input of  $D$  is labeled with a pair of the form  $\langle x : d \rangle$  with  $x \in X$  and  $d \in D$ ;
- Every other gate of  $D$  is labeled with either  $\vee$  (a  $\vee$ -gate) or  $\wedge$  (a  $\wedge$ -gate).

We denote by  $|C|$  the number of edges in  $C$ .

Given a gate  $v$  of  $C$ , the *inputs of  $v$*  are the gates  $w$  of  $C$  such that there is a directed edge from  $w$  to  $v$ . The *set of variables below  $v$* , denoted by  $\text{var}(v)$ , is then the set of variables  $x \in X$  such that there is an input  $w$  which is labeled by  $\langle x : d \rangle$  for some  $d \in D$  and which has a directed path to  $v$ . Equivalently, if  $v$  is an input labeled by  $\langle x : d \rangle$  then  $\text{var}(v) := \{x\}$ , otherwise  $\text{var}(v) := \bigcup_{i=1}^k \text{var}(v_i)$  where  $v_1, \dots, v_k$  are the inputs of  $v$ . We assume that the set  $X$  of variables of the circuit is equal to  $\text{var}(r)$  for  $r$  the output gate of  $C$ : this can be enforced without loss of generality up to removing useless variables from  $X$ .

For each gate  $v$  of  $C$ , the *set of assignments  $\text{rel}(v) \subseteq D^{\text{var}(v)}$  of  $v$*  is defined inductively as follows. If  $v$  is an input labeled by  $\langle x : d \rangle$ , then  $\text{rel}(v)$  contains only the assignment  $[x \mapsto d]$ . Otherwise, if  $v$  is an internal gate with inputs  $v_1, \dots, v_k$  then  $\text{rel}(v) := \text{rel}(v_1) \text{ op } \dots \text{ op } \text{rel}(v_k)$  where  $\text{op} \in \{\vee, \wedge\}$  is the label of  $v$ . The *set of assignments  $\text{rel}(C)$  of  $C$*  is that of its output gate. Note that, if  $D = \{0, 1\}$ , then the set of assignments of  $C$  precisely corresponds to its satisfying valuations when we see  $C$  as a Boolean circuit in the usual sense.

We say that a  $\wedge$ -gate  $v$  is *decomposable* if all its inputs are on disjoint sets of variables; formally, for every pair of inputs  $v_1 \neq v_2$  of  $v$ , we have  $\text{var}(v_1) \cap \text{var}(v_2) = \emptyset$ . A  $\vee$ -gate  $v$  is *smooth* if all its inputs have the same set of variables (so that implicit completion does not



occur); formally, for every pair of inputs  $v_1, v_2$  of  $v$ , we have  $\text{var}(v_1) = \text{var}(v_2)$ . A  $\vee$ -gate  $v$  is *deterministic* if every assignment of  $v$  is computed by only one of its inputs; formally, for every pair of inputs  $v_1 \neq v_2$  of  $v$ , if  $\tau \in \text{rel}(v)$  then either  $\tau|_{\text{var}(v_1)} \notin \text{rel}(v_1)$  or  $\tau|_{\text{var}(v_2)} \notin \text{rel}(v_2)$ .

Let  $v$  be an internal gate with inputs  $v_1, \dots, v_k$ . Observe that if  $v$  is decomposable, then  $\text{rel}(v) = \times_{i=1}^k \text{rel}(v_i)$ . If  $v$  is smooth then  $\text{rel}(v) = \cup_{i=1}^k \text{rel}(v_i)$ . If moreover  $v$  is deterministic, then  $\text{rel}(v) = \uplus_{i=1}^k \text{rel}(v_i)$ , where  $\uplus$  denotes disjoint union. Accordingly, we denote decomposable  $\wedge$ -nodes as  $\times$ -nodes, denote smooth  $\vee$ -nodes as  $\cup$ -nodes, and denote smooth deterministic  $\vee$ -nodes as  $\uplus$ -nodes.

A multivalued circuit is *decomposable* (resp., *smooth*, *deterministic*) if every  $\wedge$ -gate is decomposable (resp., every  $\vee$ -gate is smooth, every  $\vee$ -gate is deterministic). A *multivalued DNNF on domain  $D$  and variables  $X$*  is then a decomposable multivalued circuit on  $D$  and  $X$ . A *multivalued  $d$ -DNNF on domain  $D$  and variables  $X$*  is a deterministic multivalued DNNF on  $D$  and  $X$ . In all this paper, we only work with circuits that are both decomposable and smooth, i.e., smooth multivalued DNNFs. Note that smoothness can be ensured on Boolean circuits in quadratic time [34], and the same can be done on multivalued circuits.

**Ranking functions.** Our notion of ranking functions will give a score to each assignment, but to state their properties we define them on partial assignments. Formally, a *partial assignment* is a mapping  $\nu : X \rightarrow D \cup \{\perp\}$ , where  $\perp$  is a fresh symbol representing *undefined*. We denote by  $\overline{D^X}$  the set of partial assignments on domain  $D$  and variables  $X$ . The *support*  $\text{supp}(\nu)$  of  $\nu$  is the subset of  $X$  on which  $\nu$  is defined.

We extend the definitions of compatibility, of  $\bowtie$ , and of disjointness, to partial assignments in the following way. Two partial assignments  $\tau \in \overline{D^Y}$  and  $\sigma \in \overline{D^Z}$  are *compatible*, again written  $\tau \simeq \sigma$ , when for every  $x \in Y \cap Z$ , if  $\tau(x) \neq \perp$  and  $\sigma(x) \neq \perp$  then  $\tau(x) = \sigma(x)$ . In this case, we denote by  $\tau \bowtie \sigma$  the partial assignment of  $\overline{D^{Y \cup Z}}$  defined by: for  $y \in Y \setminus Z$  we have  $(\tau \bowtie \sigma)(y) := \tau(y)$ , for  $z \in Z \setminus Y$  we have  $(\tau \bowtie \sigma)(z) := \sigma(z)$ , and for  $x \in Y \cap Z$ , if  $\tau(x) \neq \perp$  then  $(\tau \bowtie \sigma)(x) = \tau(x)$ , otherwise  $(\tau \bowtie \sigma)(x) = \sigma(x)$ . We call  $\tau$  and  $\sigma$  *disjoint* if  $Y \cap Z = \emptyset$ ; then again they are always compatible and we write  $\tau \times \sigma$  for  $\tau \bowtie \sigma$ .

We then consider ranking functions defined on partial assignments  $\overline{D^X}$ , on which we will impose *subset-monotonicity*. Formally, a  $(D, X)$ -*ranking function*  $w$  is a function<sup>1</sup>  $\overline{D^X} \rightarrow \mathbb{R}$  that gives a score to every partial assignment. Such a ranking function induces a *weak ordering*<sup>2</sup>  $\preceq$  on  $\overline{D^X}$ , with  $\mu \preceq \mu'$  defined as  $w(\mu) \leq w(\mu')$ . We always assume that ranking functions can be computed efficiently, i.e., with running time that only depends on  $X$ , not  $D$ .

By a slight notational abuse, we define the score  $w(\tau)$  of partial assignment  $\tau \in \overline{D^Y}$  with  $Y \subseteq X$  by seeing  $\tau$  as a partial assignment on  $X$  which is implicitly extended by assigning  $\perp$  to every  $z \in X \setminus Y$ . Following earlier work [20, 36, 19], we then restrict our study to ranking functions that are *subset-monotone* [36]:

► **Definition 2.1.** A  $(D, X)$ -*ranking function*  $w : \overline{D^X} \rightarrow \mathbb{R}$  is *subset-monotone* if for every  $Y \subseteq X$  and partial assignments  $\tau_1, \tau_2 \in \overline{D^Y}$  such that  $w(\tau_1) \leq w(\tau_2)$ , for every partial assignment  $\sigma \in \overline{D^{X \setminus Y}}$  (so disjoint with  $\tau_1$  and  $\tau_2$ ), we have  $w(\sigma \times \tau_1) \leq w(\sigma \times \tau_2)$ .

We use in particular the following consequence of subset-monotonicity, where we call  $\tau \in \overline{D^X}$  *maximal* (or *maximum*) for  $w : \overline{D^X} \rightarrow \mathbb{R}$  when for every  $\tau' \in \overline{D^X}$  we have  $w(\tau') \leq w(\tau)$ :

<sup>1</sup> As usual, when we write  $\mathbb{R}$ , we assume a suitable representation, e.g., as floating-point numbers.

<sup>2</sup> Recall that a weak ordering  $\preceq$  on  $A$  is a total preorder on  $A$ , i.e.,  $\preceq$  is transitive and we have either  $x \preceq y$  or  $y \preceq x$  for every  $x, y \in A$ . In particular, it can be the case that two distinct elements  $x$  and  $y$  are tied, i.e.,  $x \preceq y$  and  $y \preceq x$ .

► **Lemma 2.2.** *Let  $R \subseteq \overline{D^Y}$  and  $S \subseteq \overline{D^Z}$  with  $Y \cap Z = \emptyset$ , and let  $w : \overline{D^{Y \cup Z}} \rightarrow \mathbb{R}$  be subset-monotone. If  $\tau$  is a maximal element of  $R$  and  $\sigma$  is a maximal element of  $S$  with respect to  $w$ , then  $\tau \times \sigma$  is a maximal element of  $R \wedge S$  with respect to  $w$ .*

We give a few examples of subset-monotone ranking functions. Let  $W : X \times D \rightarrow \mathbb{R}$  be a function assigning scores to singleton assignments, and define the  $(D, X)$ -ranking function  $\text{sum}_W : \overline{D^X} \rightarrow \mathbb{R}$  by  $\text{sum}_W(\tau) = \sum_{x \in X, \tau(x) \neq \perp} W(x, \tau(x))$ . Then  $\text{sum}_W$  is subset-monotone. Similarly define  $\text{max}_W : \overline{D^X} \rightarrow \mathbb{R}$  by  $\text{max}_W(\tau) = \max_{x \in X, \tau(x) \neq \perp} W(x, \tau(x))$ , or  $\text{prod}_W$  in a similar manner (with non-negative scores for singletons); then these are again subset-monotone. In particular, we can use  $\text{sum}_W$  to encode lexicographic orderings on  $\overline{D^X}$ .

**Enumeration and problem statement.** Our goal in this article is to efficiently enumerate the satisfying assignments of circuits in nonincreasing order according to a ranking function. We will in particular apply this for the ranked enumeration of the answers to MSO queries on trees, as we will explain in Section 5. We call this problem **RankEnum**. Formally, the input to **RankEnum** consists of a multivalued circuit  $C$  on domain  $D$  and variables  $X$ , and a  $(D, X)$ -ranking function  $w$  that is subset-monotone. The output to enumerate consists of all of  $\text{rel}(C)$ , without duplicates, in nonincreasing order of scores (with ties broken arbitrarily).

Formally, we work in the RAM model on words of logarithmic size [1], where memory cells can represent integers of value polynomial in the input length, and on which arithmetic operations take constant time. We will in particular allocate arrays of polynomial size in constant time, using lazy initialization [24]. We measure the performance of our algorithms in the framework of *enumeration algorithms*, where we distinguish two phases. First, in the *preprocessing phase*, the algorithm reads the input and builds internal data structures. We measure the running time of this phase as a function of the input; in general the best possible bound is *linear preprocessing*, e.g., preprocessing in  $O(|C|)$ . Second, in the *enumeration phase*, the algorithm produces the assignments, one after the other, without duplicates, and in nonincreasing order of scores; the order of assignments that are tied according to the ranking function is not specified. The *delay* is the maximal time that the enumeration phase can take to produce the next assignment, or to conclude that none are left. We measure the delay as a function of the input, as a function of the produced assignments (which each have size  $|X|$ ), and also as a function of the number of results that have been produced so far. The best delay is *output-linear delay*, i.e.,  $O(|X|)$ , which can be achieved for (non-ranked) enumeration of MSO queries on trees [7, 25, 3]. In our results, we will always fix  $|X|$  to a constant (for technical reasons explained in the next section), so the corresponding bound would be *constant delay*, but, like [12], we will not be able to achieve it. Also note that the memory usage of the enumeration phase is not bounded by the delay, but can grow as enumeration progresses.

**Brodal queues.** Similar to [12], our algorithms in this paper will use priority queues, in a specific implementation called a (*functional*) *Brodal queue* [15]. Intuitively, Brodal queues are priority queues which support union operations in  $O(1)$ , and which are *purely functional* in the sense that operations return a queue without destroying the input queue(s). More precisely, a *Brodal queue* is a data structure which stores a set of priority-data pairs of the form  $(\mathbf{p} : \text{foo}, \mathbf{d} : \text{bar})$  where  $\text{foo}$  is a real number and  $\text{bar}$  an arbitrary piece of data, supporting operations defined below. Brodal queues are *purely functional and persistent*, i.e., for any operation applied to some input Brodal queues, we obtain as output a new Brodal queue  $Q'$ , such that the input queues can still be used. Note that the structures of  $Q'$  and of

the input Brodal queues may be sharing locations in memory; this is in fact necessary, e.g., to guarantee constant-time bounds. However, this is done transparently, and both  $Q'$  and the input Brodal queues can be used afterwards<sup>3</sup>. Brodal queues support the following:

- *Initialize*, in time  $O(1)$ , which produces an empty queue;
- *Push*, in time  $O(1)$ , which adds to  $Q$  a priority-data pair;
- *Find-Max*, in time  $O(1)$ , which either indicates that  $Q$  is empty or otherwise returns some pair  $(\mathbf{p} : \text{foo}, \mathbf{d} : \text{bar})$  with  $\text{foo}$  being maximal among the priority-data pairs stored in  $Q$  (ties are broken arbitrarily);
- *Pop-Max*, in time  $O(\log(|Q|))$ , which either indicates that  $Q$  is empty or returns two values: first the pair  $p$  returned by Find-Max, second a queue storing all the pairs of  $Q$  except  $p$ ;
- *Union*, in time  $O(1)$ , which takes as input a second Brodal queue  $Q'$  and returns a queue over the elements of  $Q$  and  $Q'$ .

### 3 Ranked Enumeration for Smooth Multivalued DNNFs

In this section, we start the presentation of our technical results by giving our algorithm to solve the ranked enumeration problem for DNNFs under subset-monotone orders. This is Result 2 from the introduction, which we restate below:

► **Theorem 3.1.** *For any constant  $n \in \mathbb{N}$ , we can solve the RankEnum problem on an input smooth multivalued DNNF circuit  $C$  on domain  $D$  and variables  $X$  with  $|X| = n$  and a subset-monotone  $(D, X)$ -ranking function with no preprocessing and with delay  $O(|D| \times |C| + \log(K + 1))$ , where  $K$  is the number of assignments produced so far.*

Note how the number  $n$  of variables is assumed to be constant in the result statement. This is for a technical reason: we will need to store partial assignments in memory, but in the RAM model we can only index polynomially many memory locations [24, page 3], so we must ensure that the total number of assignments is polynomial. The circuit itself and the domain can however be arbitrarily large, following the application to MSO queries over trees studied in Section 5: the variables of the circuit will be the variables of the MSO query (which is fixed because we will work in data complexity), and the size of the circuit and that of the domain will be linear in the size of the tree (which represents the data).

Our algorithm can be seen as an instance of the Lawler-Murty [26, 29] procedure, that has been previously used to enumerate paths in DAGs in decreasing order of weight in [36]. Interestingly, the result does not require that the input circuit is deterministic. However, it is less efficient than the method presented in Section 4 where determinism is exploited.

We prove Theorem 3.1 in the rest of this section. Let us fix a smooth multivalued DNNF  $C$  on domain  $D$  and variables  $X$ , and a subset-monotone ranking function  $w: \overline{D^X} \rightarrow \mathbb{R}$ . For a partial assignment  $\tau$ , we denote by  $w_C(\tau) = \max\{w(\tau \times \sigma) \mid \sigma \in D^X \setminus \text{supp}(\tau) \text{ and } \tau \times \sigma \in \text{rel}(C)\}$  the score of the maximal completion of  $\tau$  to a satisfying assignment of  $C$  if it exists and  $w_C(\tau) = \perp$  if no such completion exists. Our algorithm relies on the following folklore observation:

► **Lemma 3.2.** *Given a partial assignment  $\tau$ , one can compute  $w_C(\tau)$  in time  $O(|C|)$ .*

<sup>3</sup> This is similar to how persistent linked lists can be modified by removing the head element or concatenating with a new head element. Such operations can run in constant time and return the modified version of the list without invalidating the original list; with both lists sharing some memory locations in a transparent fashion.

■ **Algorithm 1** Algorithm for Theorem 3.1.

---

**Data:** Smooth multivalued DNNF  $C$  with  $n$  variables, subset-monotone ranking function  $w$ .

**Result:** Enumeration of the satisfying assignments of  $C$  in nonincreasing order of scores by  $w$ .

```

1  $Q \leftarrow$  empty priority queue;
2 Push the empty assignment  $\square$  into  $Q$  with priority  $w_C(\square)$ ;
3 while  $Q$  is not empty do
4   Pop into  $\gamma$  the assignment with maximum  $w_C$ -score from  $Q$ ;
5   for  $j \leftarrow |\text{supp}(\gamma)| + 1$  to  $n$  do
6     foreach  $d \in D$  do
7       Construct  $\alpha_d = \gamma \times \langle x_j : d \rangle$ ;
8       Compute  $w_C(\alpha_d)$  using Lemma 3.2;
9     end
10     $\gamma \leftarrow \alpha_{d_0}$  such that  $w_C(\alpha_{d_0})$  is not  $\perp$  and is maximal;
11    Push into  $Q$  all  $\alpha_{d'}$  for  $d' \neq d_0$  where  $w_C(\alpha_{d'}) \neq \perp$ , with priority  $w_C(\alpha_{d'})$ ;
12  end
13  Output  $\gamma$ ;
14 end

```

---

**Proof.** Let  $X$  be the variables of  $C$ . It is enough to show that we can compute, given a smooth multivalued DNNF  $C'$  and monotone ranking function  $w'$ , some  $\sigma' \in \text{rel}(C')$  that maximizes  $w'(\sigma')$ , in  $O(|C'|)$ . Indeed, if this is the case we can first compute the conditioning<sup>4</sup>  $C'$  of  $C$  on  $\tau$  in time  $O(|C'|)$ : specifically,  $C'$  is a multivalued circuit on domain  $D$  and variables  $X \setminus \text{supp}(\tau)$  such that, for  $\sigma' \in D^{X \setminus \text{supp}(\tau)}$  we have that  $\sigma' \in \text{rel}(C')$  iff  $\tau \times \sigma' \in \text{rel}(C)$ . Then, letting  $w'$  be the ranking function on  $\overline{D^{X \setminus \text{supp}(\tau)}}$  defined by  $w'(\sigma') := w(\sigma' \times \tau)$  (which is subset-monotone), find one such  $\sigma' \in \text{rel}(C')$  in time  $O(|C'|)$ , and then return  $w(\sigma' \times \tau)$ . This is correct thanks to subset-monotonicity of  $w$ , more precisely, by Lemma 2.2.

Now the algorithm to do this proceeds by bottom-up induction as follows: for each gate  $v$  of  $C'$ , we compute  $\sigma_v \in \text{rel}(v)$  such that  $w'(\sigma_v) = \max\{w'(\sigma) \mid \sigma \in \text{rel}(v)\}$ . If  $v$  is an input then  $\text{rel}(v)$  is a singleton assignment, and we let  $\sigma_v$  be this assignment. Now, if  $v$  is a  $\times$ -gate with inputs  $v_1, \dots, v_k$ , we let  $\sigma_v = \sigma_{v_1} \times \dots \times \sigma_{v_k}$ . By Lemma 2.2,  $\sigma_v$  is maximal for  $\text{rel}(v)$  if each  $\sigma_{v_i}$  is maximal for  $\text{rel}(v_i)$  which is the case by induction. Finally, if  $v$  is a  $\cup$ -gate with input  $v_1, \dots, v_k$ , we define  $\sigma_v = \arg \max_{i=1}^k w'(\sigma_{v_i})$ , which is clearly maximal in  $\text{rel}(v) = \bigcup_{i=1}^k \text{rel}(v_i)$  if  $\sigma_{v_i}$  is maximal in  $\text{rel}(v_i)$  for each  $i$  because  $v$  is smooth, which is the case by induction. ◀

With this in place, we are ready to describe the algorithm. Notice that our definition of multivalued circuits implies that  $\text{rel}(C)$  can never be empty, because all gates except input gates have inputs, and the circuit is decomposable. We fix an arbitrary order on  $X = \{x_1, \dots, x_n\}$  and, for  $i \in \{1, \dots, n+1\}$ , we denote by  $X_{<i}$  the set  $\{x_1, \dots, x_{i-1}\}$  (which is empty for  $i = 1$ ). A partial assignment  $\tau \in \overline{D^X}$  is called a *prefix assignment* if  $\text{supp}(\tau) = X_{<i}$  for some  $i \in \{1, \dots, n+1\}$ .

---

<sup>4</sup> See [18, Definition 5.4] for the definition of conditioning on Boolean circuits, which easily adapts to multivalued circuits.

The enumeration algorithm is then illustrated as Algorithm 1, which we paraphrase in text below. The algorithm uses a variable  $\gamma$  holding a prefix assignment and a priority queue  $Q$  containing prefix assignments. The priorities in the queue are the  $w_C$ -score, i.e., the priority of each prefix assignment is the score returned by  $w_C$  on this assignment. We initialize  $Q$  to contain only the empty partial assignment (i.e., the assignment that maps every variable to  $\perp$ , denoted  $\square$  in Algorithm 1): note that the  $w_C$ -score of  $\square$  is not  $\perp$  because  $\text{rel}(C) \neq \emptyset$ . We then do the following until the queue is empty. We pop (i.e., call *Pop-Max*) from the queue a prefix assignment (of maximal  $w_C$ -score) that we assign to  $\gamma$ ; we will inductively see that  $\gamma$  is a prefix assignment of  $D^{<i}$  for some  $i \in \{1, \dots, n+1\}$  and that its  $w_C$ -score is not  $\perp$ . We then do the following for  $j := i$  to  $n$  (i.e., potentially zero times, in case  $i = n+1$  already). For every possible choice of domain element  $d \in D$ , we let  $\alpha_d$  be the prefix assignment that extends  $\gamma$  by assigning  $x_i$  to  $d$ , and we compute the value  $w_C(\alpha_d)$  using Lemma 3.2. Among these values, the definition of  $w_C$  ensures that one has a  $w_C$ -score which is not  $\perp$ , because this is true of  $\gamma$ . We thus pick a value  $d_0 \in D$  such that  $w_C(\alpha_{d_0})$  is maximal (in particular non- $\perp$ ). We set  $\gamma$  to  $\alpha_{d_0}$ , and we push into  $Q$  all other prefix assignments  $\alpha_{d'}$  for  $d' \neq d_0$  for which we have  $w_C(\alpha_{d'}) \neq \perp$ . Once we have run this for all values of  $j$ , we have  $i = n+1$ , hence  $\gamma$  is a total assignment, and we output it. We then continue processing the remaining contents of the queue.

**Correctness of the algorithm.** We can show (see the full version [2]) that the following invariants hold at the beginning and end of every while loop iteration:

1. For every  $\tau \in Q$ , no satisfying assignment of  $C$  compatible with  $\tau$  has been outputted so far;
2. For every  $\tau, \tau' \in Q$ , if  $\tau \neq \tau'$  then  $\tau \not\preceq \tau'$ ;
3. For every  $\sigma \in \text{rel}(C)$  that has not yet been outputted by the algorithm, there exists some  $i \in \{1, \dots, n+1\}$  such that  $\sigma|_{X_{<i}} \in Q$  (in fact, the previous point then implies there is at most one such  $i$ );
4. The number of elements in  $Q$  is at most  $n \times |D| \times (K+1)$ , where  $K$  is the number of assignments produced so far.

We explain next why they imply correctness.

▷ **Claim 3.3.** Algorithm 1 terminates, enumerates  $\text{rel}(C)$  without duplicates and in non-increasing order, and runs with delay  $O(|D| \times |C| + \log(K+1))$  with  $K$  the number of assignments produced so far.

*Proof.* We first show that the algorithm terminates. Indeed, notice that we pop a prefix assignment from the queue at the beginning of every while loop iteration. Let us show that, once a prefix assignment  $\tau$  has been popped from  $Q$ , it cannot be pushed again into  $Q$  for the rest of the algorithm's execution. Indeed, observe that once we pop  $\tau$  from  $Q$ , we will first push to  $Q$  assignments that are strict extensions of  $\tau$  (hence different from  $\tau$ ), and then output a satisfying assignment  $\tau'$  of  $C$  that is compatible with  $\tau$ , after which the current iteration of the while loop ends. Now, by invariant (1), no partial assignment compatible with  $\tau'$  can ever be added to  $Q$ , and in particular it is the case that  $\tau$  cannot ever be added to  $Q$ . Thus the queue becomes empty and the algorithm terminates.

Since the queue eventually becomes empty, by invariant (3), the algorithm outputs at least all of  $\text{rel}(C)$ . The fact that there are no duplicates follows from invariant (1), using a similar reasoning to how we proved termination. Furthermore, it is clear that only assignments of  $\text{rel}(C)$  are ever outputted. Therefore the algorithm indeed enumerates exactly all of  $\text{rel}(C)$  with no duplicates.

To check that assignments are enumerated in nonincreasing order, consider an iteration of the while loop where we output  $\tau \in \text{rel}(C)$ . Let  $\sigma \in \text{rel}(C)$  be an assignment that has not yet been outputted, and assume by contradiction that  $w(\tau) < w(\sigma)$ . Consider the prefix assignment  $\gamma$  that was popped from the queue  $Q$  at the beginning of that iteration; clearly by construction we have  $w_C(\gamma) = w(\tau)$ . But by invariant (3), there exists a prefix assignment  $\gamma'$  in  $Q$  of which  $\sigma$  is a completion, hence for this  $\gamma'$  we have  $w_C(\gamma') \geq w(\sigma)$  by definition of  $w_C$ , and this is strictly bigger than  $w(\gamma)$ , contradicting the fact that  $\gamma$  had maximal priority.

Last, we check that the delay between any two consecutive outputs is indeed  $O(|D| \times |C| + \log(K + 1))$ . The  $O(|D| \times |C|)$  term corresponds to the at most  $n \times |D|$  applications of Lemma 3.2 during a for loop until we produce the next satisfying assignment (remember that  $n$  is constant so it is not reflected in the delay). The  $O(\log(K + 1))$  term corresponds to the unique pop operation performed on the priority queue during a while loop iteration. Indeed, by invariant (4) the queue contains less than  $n \times |D| \times (K + 1)$  prefix assignments and the complexity of a pop operation is logarithmic in this. Since  $n$  is constant we obtain  $O(\log |D| + \log(K + 1))$ , and the  $O(\log |D|)$  gets absorbed in the  $O(|D| \times |C|)$  term.  $\triangleleft$

Thus, up to showing that the invariants hold (see the full version [2]), we have concluded the proof of Theorem 3.1.

## 4 **Ranked Enumeration for Smooth Multivalued d-DNNFs**

Having shown our polynomial-delay ranked enumeration algorithm for DNNF circuits, we move on in this section to our main technical contribution. Specifically, we present an algorithm for smooth multivalued DNNF circuits that are further assumed to be *deterministic*, but which achieves linear-time preprocessing and delay  $O(\log(K + 1))$ , where  $K$  denotes the number of satisfying assignments produced so far. This proves Result 3, which we restate below:

► **Theorem 4.1.** *For any constant  $n \in \mathbb{N}$ , we can solve the RankEnum problem on an input smooth multivalued d-DNNF circuit  $C$  with  $n$  variables and a subset-monotone ranking function, with preprocessing  $O(|C|)$  and delay  $O(\log(K + 1))$ , where  $K$  is the number of assignments produced so far.*

Let us fix for this section the set  $X$  of variables of  $C$  (with  $|X| = n$ ) and the domain  $D$ .

The rest of this section is devoted to proving Theorem 4.1. It is structured in three subsections, corresponding to the three main technical difficulties to overcome. First, we explain in Section 4.1 the preprocessing phase of the algorithm, where in particular we use Brodal queues to quickly “jump” over  $\uplus$ -gates. Second, in Section 4.2, we present a simple algorithm, that we call the  $A \odot B$  ranked enumeration algorithm, which conveys in a self-contained fashion the idea of how we handle  $\times$ -gate during the enumeration phase of the main algorithm. Last, we present the enumeration phase in Section 4.3.

### 4.1 Preprocessing Phase

The preprocessing phase is itself subdivided in four steps, described next.

**Preprocessing: first step.** We preprocess  $C$  in  $O(|C|)$  to ensure that the  $\times$ -gates of the circuit always have exactly two inputs. This can easily be done as follows. Remember that our definition of multivalued circuits does not allow  $\times$ -gates with no inputs, so this case does not occur. We can then eliminate  $\times$ -gates with one input by replacing them by their single



input. Next, we can rewrite  $\times$ -gates with more than two inputs to replace them by a tree of  $\times$ -gates with two inputs. For simplicity, let us call  $C$  again the resulting smooth multivalued d-DNNF circuit in which  $\times$ -gates always have exactly two inputs.

**Preprocessing: second step.** We compute, for every gate  $g$  of  $C$  the value  $\#g := |\text{rel}(g)|$ . This can clearly be done in linear time again, by a bottom-up traversal of  $C$  and using decomposability, determinism and smoothness. Note that  $\#g$  has value at most  $|D|^n$ , which is polynomial (as  $n$  is a constant), so this fits into one memory cell.

**Preprocessing: third step.** The third step begins by initializing for every gate  $g$  of  $C$  an empty Brodal queue  $B_g$ . We then populate those queues by a (linear-time) bottom-up traversal of the circuit, described next. This traversal will add to each queue  $B_g$  some priority-data pairs of the form  $(\mathbf{p} : w(\tau), \mathbf{d} : (g', 1, \tau))$  where  $g'$  has a (possibly empty) directed path to  $g$  and  $\tau \in \text{rel}(g)$ . We will shortly explain what is the exact content of these queues at the end of this third preprocessing step, but we already point out one invariant: once we are done processing a gate  $g$  in the traversal, then  $B_g$  contains at least one priority-data pair of this form, i.e., it is non-empty.

The traversal proceeds as follows:

- If  $g$  is an input gate labeled with  $\langle x : d \rangle$  corresponding to the singleton assignment  $\alpha = [x \mapsto d]$ , then we push into  $B_g$  the priority-data pair corresponding to this assignment:  $(\mathbf{p} : w(\alpha), \mathbf{d} : (g, 1, \alpha))$ .
- If  $g$  is a  $\times$ -gate with inputs  $g_1$  and  $g_2$  then we call *Find-Max* on the Brodal queues  $B_{g_1}$  and  $B_{g_2}$  of the inputs. These gates  $g_1$  and  $g_2$  have already been processed, so the queues  $B_{g_1}$  and  $B_{g_2}$  are non-empty, and we obtain priority-data pairs  $(\mathbf{p} : w(\tau_1), \mathbf{d} : (g'_1, 1, \tau_1))$  and  $(\mathbf{p} : w(\tau_2), \mathbf{d} : (g'_2, 1, \tau_2))$ , where  $\tau_1 \in \text{rel}(g_1)$  and  $\tau_2 \in \text{rel}(g_2)$ . We push into  $B_g$  the pair  $(\mathbf{p} : w(\tau_1 \times \tau_2), \mathbf{d} : (g, 1, \tau_1 \times \tau_2))$ .
- If  $g$  is a  $\uplus$ -gate with input gates  $g_1, \dots, g_m$  then we set  $B_g$  to be the union of  $B_{g_1}, \dots, B_{g_m}$ ; recall that the union operation on two Brodal queues can be done in  $O(1)$ , so that this union is linear in  $m$ .

It is clear that this third preprocessing step takes time  $O(|C|)$ . To describe what the queues contain at the end of this step, we need to define the notion of *exit gate* of a  $\uplus$ -gate:

► **Definition 4.2.** For a  $\uplus$ -gate  $g$  of  $C$ , an *exit gate* of  $g$  is a gate  $g'$  which is not a  $\uplus$ -gate (i.e., a  $\times$ -gate or an input of the circuit) such that there is a path from  $g'$  to  $g$  where every gate except  $g'$  on this path is a  $\uplus$ -gate. We denote by  $\text{exit}(g)$  the set of exit gates for  $g$ .

We can then characterize what the queues contain:

- ▷ **Claim 4.3.** When the third preprocessing step finishes, the queues  $B_g$  are as follows:
- If  $g$  is an input gate corresponding to the singleton assignment  $\alpha = [x \mapsto d]$  then  $B_g$  contains only the pair  $(\mathbf{p} : w(\alpha), \mathbf{d} : (g, 1, \alpha))$ .
  - If  $g$  is a  $\times$ -gate then  $B_g$  contains only one pair, which is of the form  $(\mathbf{p} : w(\tau), \mathbf{d} : (g, 1, \tau))$  where  $\tau$  is some satisfying assignment of  $g$  of maximal score (i.e., maximal among  $\text{rel}(g)$ ).
  - If  $g$  is a  $\uplus$ -gate then  $B_g$  contains exactly the following: for every exit gate  $g'$  of  $g$ , the queue  $B_g$  contains one pair of the form  $(\mathbf{p} : w(\tau), \mathbf{d} : (g', 1, \tau))$  where  $\tau$  is some satisfying assignment of  $g'$  of maximal score (i.e., maximal among  $\text{rel}(g')$ ).

This implies, in particular, that for every  $g \in C$  the queue  $B_g$  contains a pair  $(\mathbf{p} : w(\tau), \mathbf{d} : (g', 1, \tau))$  (possibly  $g' = g$ ) where  $\tau$  is a satisfying assignment of  $g$  of maximal score among  $\text{rel}(g)$ .

Proof of Claim 4.3. It is routine to prove this by bottom-up induction, in particular using Lemma 2.2 for the case of  $\times$ -gates. ◁



This concludes the third preprocessing step. Intuitively, the Brodal queues computed at this step will allow us to jump directly to the exits of  $\uplus$ -gates, without spending time traversing potentially long paths of  $\uplus$ -gates. Thanks to the constant-time union operation on Brodal queues, this third step takes linear time, and in fact this is the only part of the proof where we need this bound on the union operation. More precisely, in the remainder of the algorithm, we will only use on priority queues  $Q$  the operations *Initialize*, *Push* and *Find-Max* (in  $O(1)$ ) and *Pop-Max* (in  $O(\log |Q|)$ ).

**Preprocessing: fourth step.** In the fourth and last preprocessing step, we define some more data structures on every gate  $g$  of  $C$ .

First, we define for every gate  $g$  a priority queue  $Q_g$ . For all input gates and  $\uplus$ -gates, we simply set  $Q_g := B_g$ , but for  $\times$ -gates we will define  $Q_g$  to be new priority queues. Once this is done, we will only use the priority queues  $Q_g$ , and can forget about the priority queues  $B_g$ . We construct  $Q_g$  for each  $\times$ -gate  $g$  separately, in  $O(1)$  time, as follows. Letting  $g_1$  and  $g_2$  be the inputs to  $g$ , we call Find-Max on  $B_g$ . By Claim 4.3, we obtain a pair  $(\mathbf{p} : w(\tau), \mathbf{d} : (g, 1, \tau))$  where  $\tau$  is some satisfying assignment of  $g$  of maximal score. We split  $\tau$  into  $\tau_1 \times \tau_2$  where  $\tau_i \in \text{rel}(g_i)$  for  $i \in \{1, 2\}$ , and we define the priority queue  $Q_g$  to contain one priority-data pair, namely,  $(\mathbf{p} : w(\tau), \mathbf{d} : (1, 1, \tau_1, \tau_2))$ .

Second, we allocate for every gate  $g$  a table  $T_g$  of size  $\#g$  (indexed starting from 1), that will later hold satisfying assignments of  $g$  in nonincreasing order of scores, stored into contiguous memory cells starting at the beginning of  $T_g$ . We do not bother initializing these tables, but we initialize integers  $i_g$  to 0, that will store the current number of assignments stored in  $T_g$ .

Last, we also initialize to 0 a bidimensional bit table  $R_g$  for every  $\times$ -gate  $g$ , of size  $\#g_1 \times \#g_2$  with  $g_1, g_2$  the two inputs of  $g$ . This can be done in  $O(1)$  with the technique of *lazy initialization*, see e.g., [24, Section 2.5]. The role of these tables will be explained later.

This concludes the description of the preprocessing phase of our algorithm. In what follows, we will rely on the priority queues  $Q_g$ , the tables  $T_g$ , the integers  $i_g$  storing their size, and the tables  $R_g$ . The following should then be clear:

▷ **Claim 4.4.** Once we finish the fourth preprocessing step (concluding the preprocessing), all integers  $i_g$  are 0, all tables  $T_g$  and  $R_g$  are empty, and the queues  $Q_g$  contain the following:

- If  $g$  is an input gate corresponding to the singleton assignment  $\alpha = [x \mapsto d]$ , then  $Q_g$  contains only the pair  $(\mathbf{p} : w(\alpha), \mathbf{d} : (g, 1, \alpha))$ .
- If  $g$  is a  $\times$ -gate with inputs  $g_1, g_2$ , then  $Q_g$  contains only one priority-data pair which is of the form  $(\mathbf{p} : w(\tau_1 \times \tau_2), \mathbf{d} : (1, 1, \tau_1, \tau_2))$ , where  $\tau_1 \times \tau_2$  is some satisfying assignment of  $g$  of maximal score (among  $\text{rel}(g)$ ).
- If  $g$  is a  $\uplus$ -gate, then  $Q_g$  contains, for every exit gate  $g'$  of  $g$ , one pair of the form  $(\mathbf{p} : w(\tau), \mathbf{d} : (g', 1, \tau))$  where  $\tau$  is some satisfying assignment of  $g'$  of maximal score (among  $\text{rel}(g')$ ).

Again, this in particular implies that each  $Q_g$  stores a satisfying assignment of  $g$  of maximal score (but the way in which it is represented depends on the type of  $g$ ).

## 4.2 $A \odot B$ Ranked Enumeration Algorithm

Having described the preprocessing phase, we present in this section a component of the enumeration phase of our algorithm, called the  $A \odot B$  ranked enumeration algorithm. This simple algorithm will be used at every  $\times$ -gate  $g$  during the enumeration phase to enumerate all ways to combine the assignments of the two inputs of  $g$ .

■ **Algorithm 2** Algorithm for  $A \odot B$  ranked enumeration.

---

**Data:** Two arrays  $A, B$  of real numbers of size  $n_1, n_2$  (indexed from 1), sorted in nonincreasing order; An operation  $\odot$  as described in the main text.

**Result:** An enumeration of the pairs  $\{(i, j) \mid (i, j) \in \{1, \dots, n_1\} \times \{1, \dots, n_2\}\}$  in nonincreasing order of the score  $A[i] \odot B[j]$

- 1  $R \leftarrow$  bidimensional array of size  $n_1 \times n_2$  lazily initialized to 0;
- 2  $Q \leftarrow$  empty priority queue;
- 3 Push  $(1, 1)$  into  $Q$  with priority  $A[1] \odot B[1]$ ;
- 4  $R[1, 1] \leftarrow$  true;
- 5 **while**  $Q$  is not empty **do**
- 6     Pop into  $(i, j)$  the pair with maximal priority from  $Q$ ;
- 7     Output  $(i, j)$ ;
- 8     **for**  $(p, q) \in \{(i + 1, j), (i, j + 1)\}$  **do**
- 9         **if**  $p \leq n_1$  and  $q \leq n_2$  and  $R[p][q] = 0$  **then**
- 10             Push  $(p, q)$  into  $Q$  with priority  $A[p] \odot B[q]$ ;
- 11              $R[p][q] \leftarrow$  true;
- 12         **end**
- 13     **end**
- 14 **end**

---

Let  $\odot : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$  be an operation which is computable in  $O(1)$  and such that, for all  $a \leq a'$  and  $b \leq b'$  we have  $a \odot b \leq a' \odot b'$  (this is similar to subset-monotonicity, and is in fact equivalent; cf. the full version [2]). We explain in this section how, given as input two tables (indexed starting from 1)  $A, B$  of reals of size  $n_1, n_2$  sorted in nonincreasing order, we can enumerate the set of integer pairs  $\{(i, j) \mid (i, j) \in \{1, \dots, n_1\} \times \{1, \dots, n_2\}\}$ , in nonincreasing order of the score  $A[i] \odot B[j]$ , with  $O(1)$  preprocessing and a delay  $O(\log K)$  where  $K$  is the number of pairs outputted so far.

Intuitively, this will be applied at every  $\times$ -gate  $g$ , with  $[n_1]$  (resp.,  $[n_2]$ ) representing the satisfying valuations of the first (resp., second) input of  $g$  sorted in a nonincreasing order, as in the table  $A$  (resp.,  $B$ ).

The algorithm is shown in Algorithm 2, but we also paraphrase it in text with more explanations. We initialize a two-dimensional bit table  $R$  of size  $n_1 \times n_2$  to contain only zeroes (again using lazy initialization [24, Section 2.5]), whose role will be to remember which pairs have been seen so far, and a priority queue  $Q$  containing only the pair  $(\mathbf{p} : A[1] \odot B[1], \mathbf{d} : (1, 1))$ ; we set  $R[1, 1]$  to true because the pair  $(1, 1)$  has been seen. Then, while the queue is not empty, we do the following. We pop (call *Pop-Max*) from  $Q$ , obtaining a priority-data pair of the form  $(\mathbf{p} : A[i] \odot B[j], \mathbf{d} : (i, j))$ . We output the pair  $(i, j)$ . Then, for each  $(p, q) \in \{(i + 1, j), (i, j + 1)\}$  that is in the  $[n_1] \times [n_2]$  grid, if the pair  $(p, q)$  has not been seen before, then we push into  $Q$  the pair  $(\mathbf{p} : A[p] \odot B[q], \mathbf{d} : (p, q))$  and mark  $(p, q)$  as seen in  $R$ . We show the following in the full version [2].

▷ **Claim 4.5.** This  $A \odot B$  ranked enumeration algorithm is correct and runs with the stated complexity.

*Proof sketch.* The proof is simple and hinges on the following two invariants:

1. For any pair  $(i, j)$  not enumerated so far, there exists a pair  $(i', j')$  (possibly  $(i, j) = (i', j')$ ) such that  $(i', j')$  is in  $Q$ , and a simple path in the  $[n_1] \times [n_2]$  grid from  $(i', j')$  to  $(i, j)$  with nondecreasing first and second coordinates such that none of the pairs in that path have been outputted yet.
2. The queue contains at most  $K + 1$  pairs for  $K$  the number of pairs outputted so far. ◁

### 4.3 Enumeration Phase

We last move on to the enumeration phase. We first give a high-level description of how the enumeration phase works, before presenting the details.

**The operation  $\text{Get}(g, j)$ .** We will define a recursive operation  $\text{Get}$ , running in complexity  $O(\log(K + 1))$ , that applies to a gate  $g$  and integer  $1 \leq j \leq i_g + 1$  and does the following. If  $j \leq i_g$  then  $\text{Get}(g, j)$  simply returns the satisfying assignment of  $g$  that is stored in  $T_g[j]$  (i.e., this assignment has already been computed). Otherwise, if  $j = i_g + 1$ , then  $\text{Get}(g, j)$  finds the next assignment to be enumerated, inserts it into  $T_g$ , and returns that assignment. Note that, in this case, calling  $\text{Get}(g, j)$  modifies the memory for  $g$  and some other gates  $g'$ . Specifically, it modifies the tables  $T_{g'}$  and  $R_{g'}$ , the queues  $Q_{g'}$ , and the integers  $i_{g'}$  for various gates  $g'$  having a directed path to  $g$  (i.e., including  $g' = g$ ).

When we are not executing an operation  $\text{Get}$ , the memory will satisfy the following invariants, for every  $g$  of  $C$ :

- The table  $T_g$  contains assignments  $\tau \in \text{rel}(g)$ , ordered by nonincreasing score and with no duplicates; and  $i_g$  is the current size of  $T_g$ ;
- For any assignment  $\tau \in \text{rel}(g)$  that does not occur in  $T_g$ , it is no larger than the last assignment in  $T_g$ , i.e., we have  $w(\tau) \leq w(T_g[i_g])$ .
- The queues  $Q_g$  will also satisfy some invariants, which will be presented later.
- The tables  $R_g$  for the  $\times$ -gates record whether we have already seen pairs of satisfying assignments of the two children, similarly to how this is done in the  $A \odot B$  algorithm.

The tables  $T_g$  store the assignments in the order in which we find them, which is compatible with the ranking function. This allows us, in particular, to obtain in constant time the  $j$ -th satisfying assignment of  $\text{rel}(g)$  if it has already been computed, i.e., if  $j \leq i_g$ . The reason why we keep the assignments in the tables  $T_g$  is because we may reach the gate  $g$  via many different paths throughout the enumeration, and these paths may be at many different stages of the enumeration on  $g$ .

At the top level, if we can implement  $\text{Get}$  while satisfying the invariants above, then the enumeration phase of the algorithm is simple to describe: for  $j$  ranging from 1 to  $\#r$ , we output  $\text{Get}(r, j)$ , where  $r$  is the output gate of  $C$ .

**Implementing  $\text{Get}$ .** We first explain the intended semantics of data values in the queues  $Q_b$ :

- If  $g$  is a  $\oplus$ -gate then  $Q_b$  will always contain pairs of the form  $(\mathbf{p} : w(\tau), \mathbf{d} : (g', j, \tau))$  where  $g' \in \text{exit}(g)$  and  $j \in \{1, \dots, i_{g'} + 1\}$  and  $\tau \in \text{rel}(g')$ , and the idea is that at the end of the enumeration  $\tau$  will be stored at position  $j$  in  $T_{g'}$ .
- If  $g$  is a  $\times$ -gate, letting  $g'_1$  and  $g'_2$  be the input gates, then  $Q_b$  will always contain pairs of the form  $(\mathbf{p} : w(\tau_1 \times \tau_2), \mathbf{d} : (j_1, j_2, \tau_1, \tau_2))$  with  $\tau_i \in \text{rel}(g'_i)$  and at the end of the enumeration  $\tau_i$  will be at position  $j_i$  in  $T_{g'_i}$  with  $j_i \in \{1, \dots, i_{g'_i} + 1\}$  for all  $i \in \{1, 2\}$ .
- If  $g$  is an input gate, then  $Q_b$  initially contains the only assignment captured by  $g$ , becomes empty the first time we call  $\text{Get}(g, 1)$ , and remains empty thereafter.

The implementation of  $\text{Get}$  is given in Algorithm 3. Intuitively, the algorithm for  $\oplus$ -gates simply consists of interleaving the maximal assignments of its exit gates, similarly to how one builds a sorted list for the union of two or more sorted lists. Here, determinism ensures that we do not get duplicates. The algorithm for  $\times$ -gates proceeds similarly to the  $A \odot B$  algorithm, as explained in the previous section.

This concludes the presentation of the function  $\text{Get}$ , and with it that of the enumeration phase of the algorithm. The discussion of the delay bound can be found in the full version [2].

■ **Algorithm 3** Implementation of  $\text{Get}(g, j)$  for the enumeration phase.

---

**Data:** The tables  $T_g, R_g$ , queues  $Q_g$ , integers  $\#g, i_g$ , ranking function  $w$ , a gate  $g$ , and integer  $j \in \{1, \dots, i_g + 1\}$ .

**Result:** The  $j$ -th satisfying assignment of  $g$ .

```

1 if  $j \leq i_g$  then return  $T_g[j]$ ;
  // From now on, we have  $j = i_g + 1$ 
2 if  $g$  is an input gate then
3    $(\mathbf{p} : \delta, \mathbf{d} : (g, 1, \tau')) \leftarrow \text{Pop from } Q_j$ ;
4    $\tau \leftarrow \tau'$ ;
5 end
6 else if  $g$  is a  $\oplus$ -gate then
7    $(\mathbf{p} : \delta, \mathbf{d} : (g', j', \tau')) \leftarrow \text{Pop from } Q_j$ ;
8    $\tau \leftarrow \tau'$ ;
9   if  $j' + 1 \leq \#g'$  then
10     $\tau'' \leftarrow \text{Get}(g', j' + 1)$ ;
11    Push into  $Q_g$  the priority-data pair  $(\mathbf{p} : w(\tau''), \mathbf{d} : (g', j' + 1, \tau''))$ ;
12   end
13 end
14 else if  $g$  is a  $\times$  gate then
15    $(\mathbf{p} : \delta, \mathbf{d} : (j_1, j_2, \tau_1, \tau_2)) \leftarrow \text{Pop from } Q_j$ ;
16    $\tau \leftarrow \tau_1 \times \tau_2$ ;
17   for  $(p, q) \in \{(j_1 + 1, j_2), (j_1, j_2 + 1)\}$  do
18     if  $p \leq \#g_1$  and  $q \leq \#g_2$  and  $R_g[p][q] = \text{false}$  then
19        $\tau'_1 \leftarrow \text{Get}(g_1, p)$ ;
20        $\tau'_2 \leftarrow \text{Get}(g_2, q)$ ;
21        $\tau' \leftarrow \tau'_1 \times \tau'_2$ ;
22       Push into  $Q_g$  the priority-data pair  $(\mathbf{p} : w(\tau'), \mathbf{d} : (p, q, \tau'))$ ;
23        $R_g[p][q] \leftarrow \text{true}$ ;
24     end
25   end
26 end
27  $T_g[i_g + 1] \leftarrow \tau$ ;
28  $i_g \leftarrow i_g + 1$ ;
29 return  $\tau$ 

```

---

## 5 Application to Monadic Second-Order Queries

Having presented our results on ranked enumeration for smooth multivalued DNNFs and d-DNNFs, we present their consequences in this section for the problem of ranked enumeration of MSO query answers on trees. We first present some preliminaries on trees and MSO, formally define the evaluation problem, and explain how to reduce it to our results on circuits.

**Trees and MSO on trees.** We fix a finite set  $\Lambda$  of *tree labels*. A  $\Lambda$ -tree is then a tree  $T$  whose nodes carry a label from  $\Lambda$ , and which is rooted, ordered, binary, and full, i.e., every node has either no children (a *leaf*) or exactly one *left child* and one *right child* (an *internal node*). We often abuse notation and write  $T$  to refer to its set of nodes.

We consider *monadic second-order logic* (MSO) on trees, which extends first-order logic with quantification over sets. The signature of MSO on  $\Lambda$ -trees allows us to refer to the left child and right child relationships along with unary predicates referring to the node labels; and it can express, e.g., the set of descendants of a node. We only consider MSO queries where the free variables are first-order. We omit the precise semantics of MSO; see, e.g., [27].

Fixing an MSO query  $\Phi(x_1, \dots, x_n)$  on  $\Lambda$ -trees, given a  $\Lambda$ -tree  $T$ , the *answers* of  $\Phi$  on  $T$  are the assignments  $\alpha$  on variables  $X = \{x_1, \dots, x_n\}$  and domain  $T$  such that  $\Phi(\alpha)$  holds on  $T$  in the usual sense. It is known that, for any such query  $\Phi$ , given  $T$  and an assignment  $\alpha$  from  $X$  to  $T$ , we can check whether  $\Phi(\alpha(X))$  holds in linear time. What is more, given  $T$ , we can enumerate the answers of  $\Phi$  on  $T$  with linear preprocessing and constant delay [7, 25, 3].

We now define *ranked enumeration*. For a tree  $T$  and variables  $X = \{x_1, \dots, x_n\}$ , a  $(T, X)$ -ranking function is simply a ranking function as in Section 2, whose domain is the set of nodes of  $T$ . We still assume that ranking functions are subset-monotone. The *ranked enumeration* problem for a fixed MSO query  $\Phi$  with variables  $X$ , also denoted **RankEnum**, takes an input a tree  $T$  and a subset-monotone  $(T, X)$ -ranking function  $w$ , and must enumerate all answers of  $\Phi$  on  $T$ , without duplicates, in nonincreasing order of scores (with ties broken arbitrarily).

**Ranked enumeration for MSO.** We are now ready to restate Result 1 from the introduction:

► **Theorem 5.1.** *For any fixed tree signature  $\Lambda$  and MSO query  $\Phi$  on variables  $X$  on  $\Lambda$ -trees, given a  $\Lambda$ -tree  $T$  and a subset-monotone  $(T, X)$ -ranking function  $w$ , we can solve the **RankEnum** problem for  $\Phi$  on  $T$  and  $w$  with preprocessing time  $O(|T|)$  and delay  $O(\log(K+1))$  where  $K$  is the number of answers produced so far.*

Recall that, as the total number of answers is at most  $|T|^{|X|}$  and  $|X|$  is constant, then this implies a delay bound of  $O(\log |T|)$ . The result is simply shown by constructing a smooth multivalued d-DNNF representing the query answers. This can be done in linear time with existing techniques (we provide a self-contained proof in the full version [2]):

► **Proposition 5.2** ([3, 5]). *For any fixed tree signature  $\Lambda$  and MSO query  $\Phi$  on variables  $X$  on  $\Lambda$ -trees, given a  $\Lambda$ -tree  $T$ , we can check in time  $O(|T|)$  if  $\Phi$  has some answers on  $T$ , and if yes we can build in time  $O(|T|)$  a smooth multivalued d-DNNF  $C$  on domain  $T$  and variables  $X$  such that  $\text{rel}(C)$  is precisely the set of answers of  $\Phi$  on  $T$ .*

Note that we exclude the case where  $\Phi$  has no answer on  $T$ , because our definition of multivalued circuits does not allow them to capture an empty set of assignments; of course we can do this check in the preprocessing, and if there are no answers then enumeration is trivial.

These results are intuitively shown by translating the MSO query to a tree automaton, and then computing a provenance circuit of this automaton by a kind of product construction [6]. The resulting circuit is a smooth multivalued DNNF, and is additionally a d-DNNF if the automaton is deterministic. We can then show Theorem 5.1 simply by performing the compilation (Proposition 5.2) as part of the preprocessing, and then invoking the enumeration algorithm of Section 4 (Theorem 4.1). Notice that we could also use the algorithm of Section 3 (Theorem 3.1), in particular if it is easier to obtain a nondeterministic tree automaton for the query, as its provenance circuit would then be a non-deterministic DNNF [5].

## 6 Conclusion

We have studied the problem of ranked enumeration for tractable circuit classes from knowledge compilation, namely, DNNFs and d-DNNFs, in the setting of multivalued circuits so as to apply these results to ranked enumeration for MSO query answers on trees. We have

shown that the latter task can be solved with linear-time preprocessing and delay logarithmic in the number of answers produced so far, in particular logarithmic delay in the input tree in data complexity. This result on trees is the analogue of a previous result on words [12], achieving the same bounds but for a different notion of ranking functions.

We leave several questions open for future work. For instance, our efficient algorithms always assume that the input circuits are smooth: although this can be ensured “for free” in the setting of MSO on trees, it is generally quadratic to enforce on an arbitrary input circuit [34]. It may be possible to perform enumeration directly on non-smooth circuits, or on implicitly smoothed circuits, e.g., with special gates as in [3]. It would also be natural to study this problem in combined complexity, or for free second-order variables, though our algorithms cannot work on the RAM model if we need to store a superpolynomial number of assignments in memory. Last, it may be possible to extend our algorithms to more general ranking functions than the one we study, for instance by leveraging the framework of MSO cost functions used in [12], or using weighted logics [22], or possibly replacing subset-monotonicity by a weaker guarantee.

Last, it would be interesting to study whether our results can extend to the support of *updates*, e.g., reweighting updates to the ranking functions, or updates on the underlying circuits or (for MSO queries) on the tree, as in [28] or [5]. However, this is more difficult than the case of updates for non-ranked enumeration, because our algorithms use larger intermediate structures which are more challenging to maintain.

---

## References

- 1 Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The design and analysis of computer algorithms*. Addison-Wesley, 1974.
- 2 Antoine Amarilli, Pierre Bourhis, Florent Capelli, and Mikaël Monet. Ranked enumeration for MSO on trees via knowledge compilation. *CoRR*, abs/2310.00731, 2023. Full version of this article with all proofs. doi:10.48550/arXiv.2310.00731.
- 3 Antoine Amarilli, Pierre Bourhis, Louis Jachiet, and Stefan Mengel. A circuit-based approach to efficient enumeration. In *ICALP, 2017*. doi:10.4230/LIPIcs.ICALP.2017.111.
- 4 Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. Constant-delay enumeration for nondeterministic document spanners. In *ICDT, 2019*. doi:10.4230/LIPIcs.ICDT.2019.22.
- 5 Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. Enumeration on trees with tractable combined complexity and efficient updates. In *PODS, 2019*. doi:10.1145/3294052.3319702.
- 6 Antoine Amarilli, Pierre Bourhis, and Pierre Senellart. Provenance circuits for trees and treelike instances. In *ICALP, 2015*. doi:10.1007/978-3-662-47666-6\_5.
- 7 Guillaume Bagan. MSO queries on tree decomposable structures are computable with linear delay. In *CSL, 2006*. doi:10.1007/11874683\_11.
- 8 Guillaume Bagan, Arnaud Durand, and Étienne Grandjean. On acyclic conjunctive queries and constant delay enumeration. In *CSL, 2007*. doi:10.1007/978-3-540-74915-8\_18.
- 9 Nurzhan Bakibayev, Tomáš Kociský, Dan Olteanu, and Jakub Závodný. Aggregation and ordering in factorised databases. *PVLDB*, 6(14), 2013. doi:10.14778/2556549.2556579.
- 10 Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering UCQs under updates and in the presence of integrity constraints. In *ICDT, 2018*. doi:10.4230/LIPIcs.ICDT.2018.8.
- 11 Pierre Bourhis, Laurence Duchien, Jérémie Dusart, Emmanuel Lonca, Pierre Marquis, and Clément Quinton. Pseudo polynomial-time top-k algorithms for d-dnnf circuits. *CoRR*, 2022. doi:10.48550/arXiv.2202.05938.
- 12 Pierre Bourhis, Alejandro Grez, Louis Jachiet, and Cristian Riveros. Ranked enumeration of MSO logic on words. In *ICDT, 2021*. doi:10.4230/LIPIcs.ICDT.2021.20.
- 13 Karl Bringmann, Nofar Carmeli, and Stefan Mengel. Tight fine-grained bounds for direct access on join queries. In *PODS, 2022*. doi:10.1145/3517804.3526234.



- 14 Karl Bringmann, Nofar Carmeli, and Stefan Mengel. Tight fine-grained bounds for direct access on join queries. *CoRR*, abs/2201.02401, 2022. doi:10.48550/arXiv.2201.02401.
- 15 Gerth Stølting Brodal and Chris Okasaki. Optimal purely functional priority queues. *Journal of Functional Programming*, 6(6), 1996. doi:10.1017/S095679680000201X.
- 16 Nofar Carmeli and Markus Kröll. On the enumeration complexity of unions of conjunctive queries. *TODS*, 46(2), 2021. doi:10.1145/3450263.
- 17 Nofar Carmeli, Nikolaos Tziavelis, Wolfgang Gatterbauer, Benny Kimelfeld, and Mirek Riedewald. Tractable orders for direct access to ranked answers of conjunctive queries. *TODS*, 48(1), 2023. doi:10.1145/3578517.
- 18 Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17, 2002. doi:10.1613/jair.989.
- 19 Shaleen Deep, Xiao Hu, and Paraschos Koutris. Ranked enumeration of join queries with projections. *arXiv preprint*, 2022. doi:10.48550/arXiv.2201.05566.
- 20 Shaleen Deep and Paraschos Koutris. Ranked enumeration of conjunctive query results. *arXiv preprint*, 2019. doi:10.48550/arXiv.1902.02698.
- 21 Johannes Doleschal, Benny Kimelfeld, Wim Martens, and Liat Peterfreund. Weight annotation in information extraction. *Logical Methods in Computer Science*, 18, 2022. doi:10.46298/lmcs-18(1:21)2022.
- 22 Manfred Droste and Paul Gastin. Weighted automata and weighted logics. In *ICALP*. Springer, 2005. doi:10.1007/11523468\_42.
- 23 Arnaud Durand and Étienne Grandjean. First-order queries on structures of bounded degree are computable with constant delay. *TOCL*, 8(4), 2007. doi:10.1145/1276920.1276923.
- 24 Étienne Grandjean and Louis Jachiet. Which arithmetic operations can be performed in constant time in the RAM model with addition? *arXiv preprint*, 2022. doi:10.48550/arXiv.2206.13851.
- 25 Wojciech Kazana and Luc Segoufin. Enumeration of monadic second-order queries on trees. *TOCL*, 14(4), 2013. doi:10.1145/2528928.
- 26 Eugene L Lawler. A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem. *Management science*, 18(7), 1972.
- 27 Leonid Libkin. *Elements of finite model theory*. Springer, 2004. doi:10.1007/978-3-662-07003-1.
- 28 Katja Losemann and Wim Martens. MSO queries on trees: Enumerating answers under updates. In *CSL-LICS*, 2014. doi:10.1145/2603088.2603137.
- 29 Katta G Murty. An algorithm for ranking all the assignments in order of increasing cost. *Operations research*, 16(3), 1968. doi:10.1287/opre.16.3.682.
- 30 Dan Olteanu and Jakub Závodný. Size bounds for factorised representations of query results. *TODS*, 40(1), 2015. doi:10.1145/2656335.
- 31 Nicole Schweikardt, Luc Segoufin, and Alexandre Vigny. Enumeration for FO queries over nowhere dense graphs. *JACM*, 69(3), 2022. doi:10.1145/3517035.
- 32 Luc Segoufin. A glimpse on constant delay enumeration (invited talk). In *STACS*, 2014. URL: <https://hal.inria.fr/hal-01070893/document>, doi:10.4230/LIPIcs.STACS.2014.13.
- 33 Luc Segoufin and Alexandre Vigny. Constant delay enumeration for FO queries over databases with local bounded expansion. In *ICDT*, 2017. doi:10.4230/LIPIcs.ICDT.2017.20.
- 34 Andy Shih, Guy Van den Broeck, Paul Beame, and Antoine Amarilli. Smoothing structured decomposable circuits. In *NeurIPS*, 2019. URL: <https://proceedings.neurips.cc/paper/2019/hash/940392f5f32a7ade1cc201767cf83e31-Abstract.html>.
- 35 Nikolaos Tziavelis, Deepak Ajwani, Wolfgang Gatterbauer, Mirek Riedewald, and Xiaofeng Yang. Optimal algorithms for ranked enumeration of answers to full conjunctive queries. *PVLDB*, 13(9), 2020. doi:10.14778/3397230.3397250.
- 36 Nikolaos Tziavelis, Wolfgang Gatterbauer, and Mirek Riedewald. Any-k algorithms for enumerating ranked answers to conjunctive queries. *arXiv preprint*, 2022. doi:10.48550/arXiv.2205.05649.
- 37 Kunihiko Wasa. Enumeration of enumeration algorithms. *CoRR*, 2016. doi:10.48550/arXiv.1605.05102.