24th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems

ATMOS 2024, September 5–6, 2024, Royal Holloway, London, United Kingdom

Edited by

Paul C. Bouman
Spyros C. Kontogiannis

OASICS

Editors

Paul C. Bouman
Erasmus University Rotterdam, The Netherlands
bouman@ese.eur.nl

Spyros C. Kontogiannis
University of Patras, Greece
spyridon.kontogiannis@upatras.gr

ACM Classification 2012
Theory of computation → Design and analysis of algorithms; Mathematics of computing → Discrete mathematics; Mathematics of computing → Combinatorics; Theory of computation → Mathematical optimization; Mathematics of computing → Graph theory; Applied computing → Transportation

ISBN 978-3-95977-350-8

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-350-8.

Publication date
October, 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.

Digital Object Identifier: 10.4230/OASIcs.ATMOS.2024.0

OASIcs – OpenAccess Series in Informatics

ISSN 1868-8969

https://www.dagstuhl.de/oasics

Contents

Papers

24th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2024).
Editors: Paul C. Bouman and Spyros C. Kontogiannis
OASICS OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

In the ongoing desire to improve, adapt and understand transportation systems, algorithmic theory and mathematical techniques are essential to develop and solve new models and optimization problems. Scientific advancements come from various disciplines, including mathematical optimization, theoretical computer science, algorithmics, and operations research. Since the 2000's, the *Algorithmic Approaches for Transportation Modelling, Optimization and Systems* (ATMOS) symposia represent a well-established series of meetings that brings together researchers and practitioners who are interested in all aspects of algorithmic methods and models for transportation optimization, providing a forum for the exchange and dissemination of new ideas and techniques to handle all modes of transportation.

The 24th edition of the *Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems* (ATMOS 2024) was held on September 5-6 2024, as part of ALGO 2024, the major annual European event for researchers, students and practitioners in algorithms, which was hosted by the Royal Holloway, University of London in Egham, United Kingdom. All optimization problems, models and algorithmic techniques which are related to transportation systems were regarded as topics of interest for ATMOS 2024, including but not limited to: charging schemes; electromobility; fairness in schedules; fare structure design; infrastructure assignment and maintenance; passenger flows; resource-constrained shortest paths; route planning; (rolling-stock, shift, vehicle, periodic-event) scheduling; time-dependent travel-times; traffic-assignment; equilibria with side-constraints; vehicle routing and dial-a-ride problems. Of particular interest were the following techniques which were used by the papers in this year's edition of ATMOS: (exact, approximate, heuristic, local-search) algorithms for transport optimization; algorithmic engineering; algorithmic game theory; Bayesian inference; labelling and indexing schemes; learning and prediction techniques; mathematical programming; multi-objective optimization; stochastic simulation.

We received a total of thirty-two submissions from all over the world, twenty-eight of them being regular papers, and the other four being short papers. All manuscripts were reviewed by at least three PC members and were evaluated on originality, technical quality, and relevance to the topics of the symposium. Based on the reviews, the program committee selected eighteen submissions (sixteen regular papers and two short papers) to be presented at the symposium. Altogether, they quite remarkably demonstrate the wide applicability of algorithmic optimization on transportation problems. In addition, Eduardo Uchoa (Universidade Federal Fluminense, Niteroi, Brazil) kindly agreed to complement the program with an invited talk "*Exact Algorithms for Vehicle Routing: advances, challenges, and perspectives*", that was presented as one of the keynote talks of ALGO 2024.

We would like to thank: the Steering Committee of ATMOS for giving us the opportunity to serve as Program Chairs of ATMOS 2024; all the authors who submitted their papers; the members of the ATMOS 2024 Program Committee and the sub-reviewers for their valuable work in evaluating all the submissions and selecting the papers appearing in this volume; Eduardo Uchoa for accepting our invitation to present an invited talk; Argyrios Deligkas and Eduard Eiben (co-chairs of the ALGO 2024 Organizing Committee) and their team at the Royal Holloway, University of London for hosting the symposium as part of ALGO 2024. We would also like to acknowledge the use of the EasyChair system for the great help in managing the submission and review processes, and Schloss Dagstuhl for publishing the proceedings of ATMOS 2024 in its OASIcs series.

24th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2024).
Editors: Paul C. Bouman and Spyros C. Kontogiannis
OASICS OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Finally, we are pleased to announce that, based on the Program Committee's reviews and decisions, the authors Justine Cauvi, Ruoying Li and Sabine Storandt have been awarded this year's "Best Paper Award of ATMOS 2024" for their paper "*Landmark Hub Labeling: Improved Bounds and Faster Query Answering*".

August 2024

Paul Bouman and Spyros Kontogiannis

Committees

Program committee chairs

- Paul Bouman (co-chair), Erasmus University Rotterdam, Netherlands
- Spyros Kontogiannis (co-chair), University of Patras, Greece

Program committee members

- David Coudert, INRIA and Université Côte d'Azur, France
- Mattia D'Emidio, University of L'Aquila, Italy
- Julian Dibbelt, Apple, USA
- Twan Dollevoet, Erasmus University Rotterdam, The Netherlands
- Daniele Frigioni, University of Aquila, Italy
- Loukas Georgiadis, University of Ioannina, Greece
- Marc Goerigk, University of Passau, Germany
- Andrew Goldberg, Amazon, USA
- Vera Grafe, University of Kaiserslautern-Landau, Germany
- Irene Heinrich, TU Darmstadt, Germany
- Jesper Larsen, Technical Universtity of Denmark, Denmark
- Niels Lindner, Zuse Institute Berlin, Germany
- Philine Schiewe, Aalto University, Finland
- Sabine Storandt, University of Konstanz, Germany
- Rolf van Lieshout, Eindhoven University of Technology, The Netherlands

Steering committee

- Alberto Marchetti-Spaccamela, Sapienza Università di Roma, Italy
- Marie Schmidt, Universität Würzburg, Germany
- Anita Schöbel, University of Kaiserslautern-Landau, Germany
- Christos Zaroliagis, University of Patras, Greece (Chair)

Organizing committee

- Karen Burnett, Royal Holloway, University of London
- Argyrios Deligkas, Royal Holloway, University of London (co-chair)
- Eduard Eiben, Royal Holloway, University of London (co-chair)
- Francisco Ferreira Ruiz, Royal Holloway, University of London
- Tiger-Lily Goldsmith, Royal Holloway, University of London
- Matthew Hague, Royal Holloway, University of London
- Reuben Rowe, Royal Holloway, University of London
- Magnus Wahlström, Royal Holloway, University of London

List of subreviewers

- Oliver Bachtler
- Sofia Brenner
- Serafino Cicerone
- Andrea D'Ascenzo
- Esmaeil Delfaraz
- Christoph Hertrich
- Eda Kaja
- Stefano Leucci
- Ruoying Li
- Berenike Masing
- Georg Schindling
- Lena Volk

List of Authors

Barbara M. Anthony (8)
Southwestern University, Georgetown, TX, USA

Ralf Borndörfer (11, 13)
Zuse Institute Berlin, Germany

Enrico Bortoletto (4)
Zuse Institute Berlin, Germany

Justine Cauvi (1)
École Normale Supérieure de Lyon, France;
Department of Computer and Information
Science, University of Konstanz, Germany

Christine Chung (8)
Connecticut College, New London, CT, USA

David Coudert (2)
Université Côte d'Azur, Inria, I3S, CNRS,
Sophia Antipolis, France

Gianlorenzo D'Angelo (7)
Gran Sasso Science Institute, L'Aquila, Italy

Andrea D'Ascenzo (2)
Luiss University, Rome, Italy

Mattia D'Emidio (2, 7)
University of L'Aquila, Italy

Ananya Das (8)
Middlebury College, VT, USA

Esmaeil Delfaraz (7)
Dept. of Information Engineering, Computer
Science and Mathematics, University of
L'Aquila, L'Aquila, Italy

Gabriele Di Stefano (7)
Dept. of Information Engineering, Computer
Science and Mathematics, University of
L'Aquila, L'Aquila, Italy

Jenny Enerbäck (3)
Department of Mathematics,
Linköping University, Sweden;
Scania CV AB, Södertälje, Sweden

Stefan Engels (12)
Chair for Design Automation,
Technical University of Munich, Germany

Lukas Eveborn (3)
Department of Mathematics,
Linköping University, Sweden

Héloïse Gachet (5)
École nationale des ponts et chaussées,
Champs-sur-Marne, France;
SNCF, DTIPG, Saint-Denis, France

Tobias Harks (17)
University of Passau, Germany

Loïc Hélouët (16)
Univ. Rennes, IRISA, CNRS & INRIA,
Rennes, France

Sven Jäger (10, 17)
RPTU Kaiserslautern-Landau, Germany

Spyros Kontogiannis (9)
Computer Engineering and Informatics
Department, University of Patras, Greece;
Computer Technology Institute and Press
"Diophantus", Patras, Greece

Ruoying Li (1)
Department of Computer and Information
Science, University of Konstanz, Germany

Tomas Lidén (6)
Department of Science and Technology (ITN),
Linköping University, Sweden

Niels Lindner (4)
Zuse Institute Berlin, Germany

Andreas Löbel (11)
IVU Traffic Technologies AG, Berlin, Germany

Fabian Löbel (11)
Zuse Institute Berlin, Germany

Michael Markl (17)
University of Passau, Germany

Berenike Masing (4)
Zuse Institute Berlin, Germany

Frédéric Meunier (5)
École nationale des ponts et chaussées,
Champs-sur-Marne, France

Matthias Müller-Hannemann (18)
Martin-Luther-Universität Halle-Wittenberg,
Germany

Andreas Paraskevopoulos (9)
Computer Technology Institute and Press
"Diophantus", Greece

24th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2024).
Editors: Paul C. Bouman and Spyros C. Kontogiannis
OASICS OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Julian Patzner (18)
Martin-Luther-Universität Halle-Wittenberg, Germany

Felix Prause (13)
Zuse Institute Berlin, Germany

Kendra Reiter (14)
Department of Computer Science, University of Würzburg, Germany

Sarah Roth (10)
RPTU Kaiserslautern-Landau, Germany

Elina Rönnberg (3)
Department of Mathematics, Linköping University, Sweden

Kenza Saiah (16)
Alstom Transport, Saint-Ouen, France

Philine Schiewe (15, 17)
Department of Mathematics and Systems Analysis, Aalto University, Finland

Christiane Schmidt (6)
Department of Science and Technology (ITN), Linköping University, Sweden

Marie Schmidt (14)
Department of Computer Science, University of Würzburg, Germany

Anita Schöbel (10, 15)
RPTU Kaiserslautern-Landau, Germany; Fraunhofer Institute for Industrial Mathematics, Kaiserslautern, Germany

Michael Stiglmayr (14)
Department of Mathematics and Informatics, University of Wuppertal, Germany

Sabine Storandt (1)
Department of Computer and Information Science, University of Konstanz, Germany

Antoine Thébault (16)
Univ. Rennes, IRISA, CNRS & INRIA, Rennes, France; Alstom Transport, Saint-Ouen, France

Reena Urban (15)
Department of Mathematics, University of Kaiserslautern-Landau (RPTU), Germany

Rolf Nelson van Lieshout (4)
Eindhoven University of Technology, The Netherlands

Steffen Weider (11)
IVU Traffic Technologies AG, Berlin, Germany

Robert Wille (12)
Chair for Design Automation, Technical University of Munich, Germany; Software Competence Center Hagenberg GmbH (SCCH), Austria

David Yuen (8)
Independent Researcher, Kapolei, HI, USA

Rabii Zahir (6)
Department of Science and Technology (ITN), Linköping University, Sweden

Christos Zaroliagis (9)
Computer Engineering and Informatics Department, University of Patras, Greece; Computer Technology Institute and Press "Diophantus", Patras, Greece

Landmark Hub Labeling: Improved Bounds and Faster Query Answering

Justine Cauvi
École Normale Supérieure de Lyon, France
Department of Computer and Information Science, University of Konstanz, Germany

Ruoying Li
Department of Computer and Information Science, University of Konstanz, Germany

Sabine Storandt
Department of Computer and Information Science, University of Konstanz, Germany

Abstract

Hub Labeling (HL) is a state-of-the-art method for answering shortest-distance queries between node pairs in weighted graphs. It provides very fast query times but also requires considerable additional space to store the label information. Recently, a generalization of HL, called Landmark Hub Labeling (LHL), has been proposed, that conceptionally allows a storage of fewer label information without compromising the optimality of the query result. However, query answering with LHL was shown to be slower than with HL, both in theory and practice. Furthermore, it was not clear whether there are graphs with a substantial space reduction when using LHL instead of HL. In this paper, we describe a new way of storing label information of an LHL such that query times are significantly reduced and then asymptotically match those of HL. Thus, we alleviate the so far greatest shortcoming of LHL compared to HL. Moreover, we show that for the practically relevant hierarchical versions (HHL and HLHL), there are graphs in which the label size of an optimal HLHL is a factor of $\Theta(\sqrt{n})$ smaller than that of an optimal HHL. We establish further novel bounds between different labeling variants. Additionally, we provide a comparative experimental study between approximation algorithms for HL and LHL. We demonstrate that label sizes in an LHL are consistently smaller than those of HL across diverse benchmark graphs, including road networks.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases Route Planning, Shortest Path, Hub Labeling

Digital Object Identifier 10.4230/OASIcs.ATMOS.2024.1

1 Introduction

Efficiently determining the shortest path distance between node pairs in a given weighted graph $G(V, E, c)$ is important for a multitude of applications, including navigation, vehicle routing, and network design [3, 4, 13, 14, 15]. Using Dijkstra's algorithm, such distance queries can be answered in $\mathcal{O}(n \log n + m)$ with $|V| = n$ and $|E| = m$. However, if many queries need to be answered on the same input graph, preprocessing methods can help to tremendously reduce subsequent query times. One state-of-the-art method is Hub Labeling (HL) [6]. In a HL, each node $v \in V$ gets assigned a node label $L(v) \subseteq V$, and the shortest path cost $c(\pi(v, w))$ is stored along for each $w \in L(v)$. The labels need to fulfill the so called cover property that demands that $\forall s, t \in V : \exists v \in L(s) \cap L(t) \cap \pi(s, t)$. In other words, for each shortest path, its end nodes need to have at least one common node from that path in their respective labels. The node is then called a hub for s, t.

This allows for a very simple and fast query answering routine: For each $v \in L(s) \cap L(t)$ compute $c(\pi(s, v)) + c(\pi(t, v))$, with both values being precomputed, and keep track of the minimum. The resulting value is guaranteed to coincide with the shortest path distance

24th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2024).
Editors: Paul C. Bouman and Spyros C. Kontogiannis; Article No. 1; pp. 1:1–1:17
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

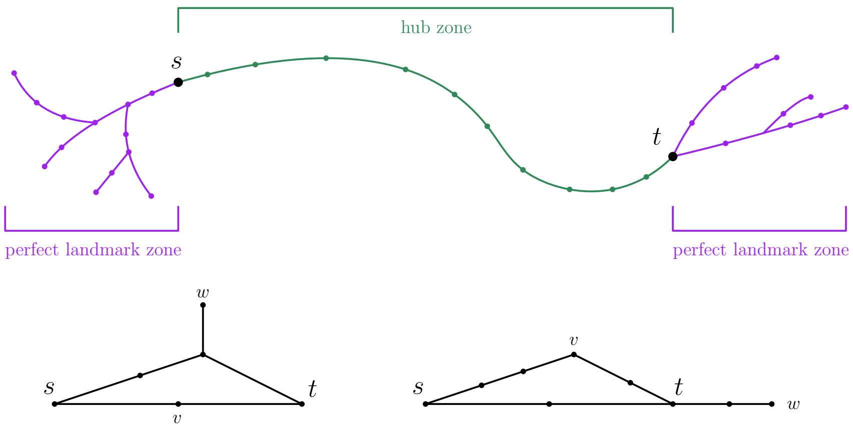


Figure 1 Top: Illustration of hub zone (green) and perfect landmark zone (purple) for the node pair s, t. Bottom: Two LHL query answering scenarios for s, t with $L(s) \cap L(t) = \{v, w\}$: In the left image (assuming uniform edge weights), the best lower bound is realized via w with $c(\pi(s, w)) - c(\pi(t, w)) = 3 - 2 = 1$ and the best upper bound is realized via v with $c(\pi(s, v)) + c(\pi(t, v)) = 1 + 1 = 2$. Here, the upper bound is tight. In the right image, the best upper bound via v is $3 + 2 = 5$ and the best lower bound via w is $4 - 2 = 2$. Here, the lower bound is tight.

between s and t. If the labels are stored sorted by ID of the contained nodes, the intersection of two labels can be computed in linear time. Thus, query answering only takes $\mathcal{O}(|L(s)| + |L(t)|) \in \mathcal{O}(L_{\max})$ where $L_{\max} := \max_{v \in V} |L(v)|$.

While HL allows for fast query answering, storing labels that meet the cover property often requires a substantial amount of space. Landmark Hub Labeling (LHL) is a generalization of HL which uses a weaker cover property and thus potentially allows for smaller label sizes [16]. The underlying observation is that if s and t both store the distance to a node v that is not on the shortest path from s to t but on a shortest path from a to b that contains both s and t, one can still deduce the correct shortest path distance between s and t, now by computing $|c(\pi(s, v)) - c(\pi(t, v))|$. Such a node v where $|c(\pi(s, v)) - c(\pi(t, v))| = c(\pi(s, t))$ is also called a perfect landmark for s, t. The relaxed cover property of LHL now demands that for each $s, t \in V$ their label intersection $L(s) \cap L(t)$ either contains a hub or a perfect landmark for s, t. The issue is that query answering becomes more intricate with LHL, as one needs to figure out whether a hub or a perfect landmark leads to the tight distance bound. To accomplish this, each node in $I := L(s) \cap L(t)$ is considered in both roles. This results in an upper bound $UB := \min_{v \in I} c(\pi(s, v)) + c(\pi(t, v))$ as well as a lower bound $LB := \max_{v \in I} |c(\pi(s, v)) - c(\pi(t, v))|$ on the shortest path distance $c(\pi(s, t))$ where either UB or LB needs to be tight, see Figure 1 for an illustration. To check if LB is tight, one needs to reconstruct the path from (w.l.o.g.) s to v up to a distance of LB and see whether t is found. If this is the case, $c(\pi(s, t)) = LB$ follows, otherwise it is certified that $c(\pi(s, t)) = UB$. To make the path traversal step efficient, a variant of LHL called path-consistent LHL (PC-LHL) was introduced in [16]. It allows to reconstruct the shortest path from a node s to one of its stored label nodes v in $\mathcal{O}(k)$ where k denotes the number on edges on that shortest path. However, this still results in distance query times of $\mathcal{O}(L_{\max} + D)$ where D denotes the diameter of the input graph. Depending on the structure of the input graph, the query time might thus be significantly larger than the $\mathcal{O}(L_{\max})$ query time of HL.

In this paper, we show that one can actually answer distance queries with a PC-LHL also in $\mathcal{O}(L_{\max})$ by storing and indexing the labels in a novel manner. Furthermore, we show that both in theory and practice substantial space savings are possible when imposing the weaker cover property of LHL instead of the stronger one required for HL.

1.1 Related Work

Hub Labeling (HL) has been studied extensively, both from a theoretical and practical perspective. Constructing a Hub Labeling (HL) with minimum total or average label size poses an NP-hard problem [2]. Interestingly, the complexity of computing a HL with minimum average label size is still open on trees. For this special case, a PTAS was described in [1]. In general graphs, an approximation algorithm with a factor of $\mathcal{O}(\log n)$ and a running time of $\mathcal{O}(n^3 \log n)$ is known [2, 8], as well as an $\mathcal{O}(\log D)$ approximation algorithm which is based on a LP-relaxation [1]. For practical applications, however, oftentimes fast heuristics are used instead of the slower and more intricate approximation algorithms. Usually, those construct a special type of HL, called a hierarchical HL (HHL) [7, 8, 12]. Here, nodes are ranked by some notion of importance and the node label of a node v can only contain nodes that have at least the same rank as v. Approximation algorithms were also studied for HHL, with the best known guarantee being in $\mathcal{O}(\sqrt{n} \log n)$ [2].

As storing the labels of a HL or a HHL requires a lot of memory, label compression techniques have been investigated [9]. Recently, Landmark Hub Labeling (LHL) has been introduced as a complementary way to reduce the label size [16]. It was shown that the $\mathcal{O}(\log n)$ approximation algorithm for HL can be adapted to also work for LHL. Furthermore, a $\mathcal{O}(\log n)$ approximation algorithm with a running time of $\mathcal{O}(n^6)$ was proposed for the path-consistent variant (PC-LHL) as well as a heuristic construction methods that run in $\mathcal{O}(n^3)$ [16, 17]. However, the query time analysis as well as a proof-of-concept-study revealed worse query times when using the PC-LHL framework instead of HL.

1.2 Contribution

In this paper, we provide numerous new structural insights into LHL.

We first carefully study the relationships between different LHL and HL variants. For example, we prove that for the hierarchical variants (HLHL and HHL), the optimal label size of the HLHL is always smaller than that of a HHL and there exist graphs where the factor between the respective label sizes is $\Theta(\sqrt{n})$. This label size gap even applies to HHL and PC-HLHL, where PC-HLHL denotes the practically relevant path-consistent HLHL variant. We show several further novel gaps and bounds, see Figure 2 for an overview of results on tree graphs, and Figure 4 for results on general graphs. Our new gaps illustrate the great potential of reducing the space consumption when using LHL instead of HL.

Next, we prove that computing an optimal PC-HLHL is NP-hard by reduction from HHL, which itself was proven to be NP-hard in [2]. To nevertheless see whether the theoretical space improvements we show for (PC-H)LHL over (H)HL also translate into practical improvements, we describe how to efficiently implement known and novel (approximation) algorithms. In an experimental study on diverse benchmark graphs, we demonstrate that label sizes of LHL are indeed consistently smaller than those of HL.

Finally, we tackle the most pronounced weakness of LHL with respect to HL, which is the query time of $\mathcal{O}(L_{\max} + D)$. The $+D$ term stems from the necessity to conduct a path traversal in the LHL query in order to check whether the computed lower or upper distance bound is tight. We describe a new method for storing labels of a (PC-)LHL such that no path traversal is needed anymore for this check. Accordingly, we can now match the $\mathcal{O}(L_{\max})$ query time of HL. We thus establish LHL as a viable alternative to HL, both with respect to space consumption and query time.

2 Preliminaries

Throughout the paper, we assume to be given an undirected graph $G = (V, E, c)$ with positive edge costs $c : E \to \mathbb{R}^+$ and a unique shortest path between any pair of nodes $s, t \in V$. The latter property can be ensured via symbolic perturbation of the edge costs. We use $\pi(s, t)$ to denote the shortest path between nodes $s, t \in V$ and $c(\pi(s, t)) = c(\pi(t, s))$ for its cost.

We already defined Hub labeling (HL) and Landmark Hub labeling (LHL) in the introduction. We now provide additional definitions for the labeling variants which are also studied in the remainder of the paper.

▶ **Definition 1** (Hierarchical Hub Labeling). *A HL L is a Hierarchical Hub Labeling (HHL) if there is a ranking $r : V \to \{1, \dots n\}$, with $n = |V|$, such that for every $v \in V$, all $w \in L(v)$ are such that $r(w) \geq r(v)$.*

For a fixed ranking r, one can find in polynomial time the HHL with minimum total label size which is called the canonical HHL.

▶ **Definition 2** (Canonical HHL). *Given a node ranking r, the canonical HHL is such that, for $v, w \in V$, $w \in L(v)$ if and only if there is a shortest path starting at v on which w has the highest rank.*

We can also define a hierarchical LHL (HLHL) in the same way as HHL by allowing all nodes to contain only nodes of higher or equal rank in their label. Another important property on labelings that we will explore is path-consistency.

▶ **Definition 3** (Path-Consistent Labeling). *A labeling is path-consistent (PC) if for each $w \in L(v)$, we also have $w \in L(u)$ for all u on a shortest path from v to w.*

In what follows, we will use PC-(L)HL to refer to a path-consistent (Landmark) Hub Labeling and PC-H(L)HL to denote a path-consistent hierarchical (Landmark) Hub Labeling.

It was shown in [16] that every canonical HHL is path-consistent. Given a ranking r, we can also define the canonical PC-HLHL, which is the PC-HLHL respecting r with the smallest labeling size [16].

▶ **Definition 4** (Canonical PC-HLHL). *Given a node ranking r, the canonical PC-HLHL is such that, for $v, w \in V$, $w \in L(v)$ if and only if there is a maximal shortest path containing v and w on which w has the highest rank.*

Finally, also the concept of a Landmark Labeling (LL) was defined in [16].

▶ **Definition 5** (Landmark Labeling). *A Landmark Labeling (LL) is a labeling $L : V \to 2^V$ which satisfies the landmark cover property, that is for every pair of nodes $s, t \in V$, there is a node $v \in L(s) \cap L(t)$ which is a perfect landmark for s, t.*

Clearly, LL is a special type of LHL just like HL.

3 Relationships between Labeling Variants

With the plethora of labeling variants that have been proposed in the literature, it is interesting to investigate how optimal label sizes behave with respect to each other. Obviously, label sizes of more general variants are at most as large as those of the respective special cases. But we are also interested in the size of the gaps that can arise between them.

In this section, we provide novel results on trees and general graphs, and summarize the known gaps between different LL, LHL, and HL classes in relationship diagrams.

3.1 On Trees

In [1], the authors show that the optimal HL on trees is hierarchical. Based on the same idea we can show that the optimal PC-LHL on trees is also hierarchical.

▶ **Theorem 6.** *For every tree $T = (V, E)$ and any valid PC-LHL L for T, there exists a hierarchical PC-LHL L' such that $|L'| \leq |L|$.*

Proof. The idea is to construct a hierarchical PC-LHL L' based on the given L without increasing the label size.

For each $u \in V$, let $L(u)$ be the label set of u, we define an induced subtree $T_u \subseteq T$ in the same way as in [1]: Let u be the root of T_u, there is a path $\pi(u, v) \in T_u$ if $v \in L(u)$. A vertex w belongs to T_u if there is a label $v \in L(u)$ such that $w \in \pi(u, v)$. We call this tree the label tree of u on T respects $L(u)$.

We now show that $T_u \cap T_v \neq \emptyset$, for every $u, v \in V$. Let $w \in L(u) \cap L(v)$, then $w \in T_u$ and $w \in T_v$. Due to *Helly Property* ([10],[11]), given a family of subtrees of T, if every two subtrees in the family intersect, then all subtrees intersect on at least one node r. To construct L', we first let $L' = L$. Secondly, we assign the node r the highest rank and add r into the label set L' of every node, then remove the labels $w \in L'(u)$ for $u \in T$, such that $r \in \pi(u, w)$. Now L' is still a valid PC-LHL with $|L'| \leq |L|$.

Let $T_1, \ldots, T_k$ be the subtrees rooted at the neighbors of r. The shortest paths between u, v are covered by r, where $u \in T_i, v \in T_j, i, j \in \{1, \ldots, k\}, i \neq j$. Therefore, we can deal with the subtrees $T_1, \ldots, T_k$ separately. Let T_i be one subtree rooted at the neighbor of r. Let $\pi(r, w)$ be the degree two path such that any node on the path, except r and w, has degree two, and the degree of w is not two. Let $u \in \pi(r, w) \setminus \{r, w\}$, then the shortest path from any node in T can be covered by r. We, therefore, remove the labels in $v \in L(u)$ such that $v \neq r$ and $v \neq u$. Now L' is still valid and $|L'| \leq |L|$. If T_i for $i \in \{1, \ldots, k\}$ is a path, we assign all nodes in T_i the lowest ranks, then $L'[T_i]$ is a valid hierarchical PC-LHL.

We obtain the subtrees that are not just a path. Let T' be a subtree after removing the degree 2 chain from r. Let $l_1, \ldots, l_p$ be the leaves in T'. Then $r \notin \pi(l_i, l_j)$ for $i \neq j$. There exists a node $u \in L(l_i) \cap L(l_j)$, and $u \in L(w)$ for all $w \in \pi(l_i, l_j)$. This means all nodes that lie on the path of two leaves contain a common label. Hence, as a pair of nodes always lies on a path from a leaf to another leaf, for any $u, v \in T'$, $L(u) \cap L(v) \neq \emptyset$. Therefore, the label trees $T'_u, T'_v \subseteq T'$ have an intersection. Again according to the *Helly Property*, all label trees have a common intersection. We assign the node the next highest rank.

For each subtree, we can apply this construction recursively, and the resulting L' is hierarchical and PC with $|L'| \leq |L|$. ◀

Since the optimal HHL is canonical and thus also PC, HHL is a special case of PC-HL. Hence, the optimal labeling of PC-HL on trees has the same size as HHL. It thus follows that it also has the same size as HL.

In Figure 2, we present the known optimal label size relations between different labelings on trees, and the blue color highlights the relations of our contribution in this work. On trees, the optimal LHL has the smallest size of any labeling. For a star graph with n nodes, the average label size in optimal LL is $\mathcal{O}(n)$, where in the optimal LHL it is constant. For a path, the optimal solution of HL has a maximum label size of $\mathcal{O}(\log n)$, while the optimal solution of LHL has a maximum size of $\mathcal{O}(1)$. The HLHL is a special labeling of LHL, but whether the optimal LHL on trees is hierarchical is still open. As we have shown that the optimal PC-LHL on trees is hierarchical and PC-HLHL is a special case of HLHL, we conclude that the optimal HLHL is at most as large as PC-LHL, PC-HLHL, and canonical PC-HLHL.

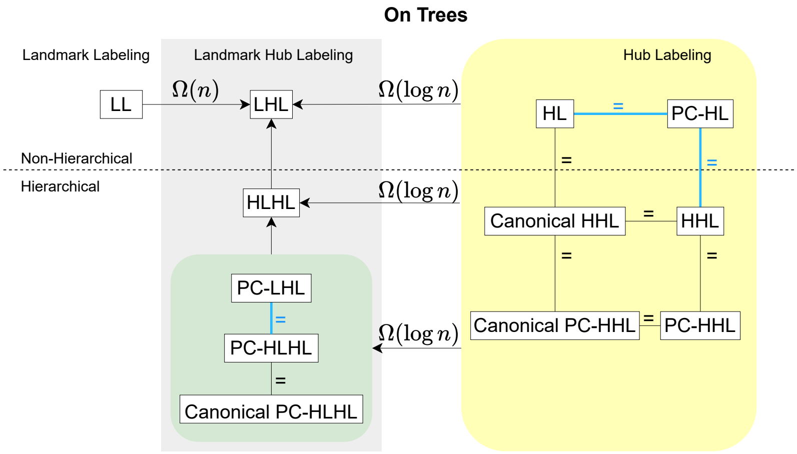


Figure 2 Relation of optimal solution between different labelings on trees. This diagram consists of three labelings, LL, LHL, and HL. In particular, we present the relations for optimal solutions on hierarchical and PC labeling of LHL and HL. The arrows point to the labeling whose optimal solution is always smaller or equal to the other one. The function classes on the arrow are the largest known factors of the optimal label size of the larger labeling to the smaller labeling on particular tree families. Equal means that both labelings have the same optimal label size on trees. The blue highlighted relations are our contributions.

3.2 On General Graphs

In [2] it was shown that the gap between HHL and HL can be in $\Omega(\sqrt{n})$. We prove that the optimal HL on the same instance satisfies the PC property. Thus, the gap between HHL to both PC-HL and PC-LHL is also at least a factor of $\Omega(\sqrt{n})$. Using the same example, we show that the gap between hierarchical and non-hierarchical (PC-)LHL is also at least a factor of $\Omega(\sqrt{n})$.

▶ **Theorem 7.** *There is a graph family for which the optimal HLHL size is $\Omega(\sqrt{n})$ times larger than the optimal label size of LHL.*

Proof. We take the same example as in [2] (that is the graph in Figure 3 without the b_i nodes) in which they prove that the gap between HL and HHL can be $\Omega(\sqrt{n})$. They exhibit a HL for the considered graph of total labeling size in $\mathcal{O}(n)$. Each node has to contain at least one label in the label set, thus the smallest summed label size of a graph without isolated nodes is in $\mathcal{O}(n)$. Since HL is also an LHL, the optimal LHL size is also in $\mathcal{O}(n)$. We show that the total size of any HLHL is $\Omega(n^{\frac{3}{2}})$. Now we consider a ranking of the nodes on the clique, let $r(c_1) < ... < r(c_k)$, and consider p_{ix} and c_j for $i < j$ and $1 \leq x \leq k$. The only shortest path containing p_{ix} and c_j is (p_{ix}, c_i, c_j). Since c_i cannot be in the label of c_j, then either p_{ix} is in the label of c_j or c_j is in the label of p_{ix}. This produces at least one label for each such pair. The total number of such pairs is:

$$k \sum_{i=1}^{k-1} i = \frac{k^2(k-1)}{2} \in \Omega(n^{\frac{3}{2}}).$$

Then the total label size of HLHL is $\Omega(n^{\frac{3}{2}})$. Hence, the gap between the optimal HLHL and the optimal LHL is $\Omega(\sqrt{n})$. ◀

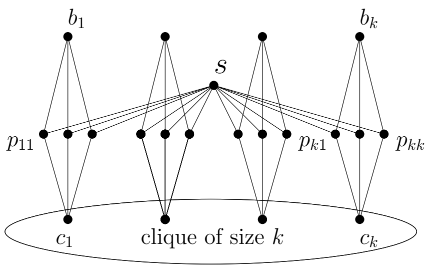


Figure 3 Example graph with $\Theta(\sqrt{n})$ label size gap between HHL and HLHL.

Now consider the gap between PC-HLHL and PC-LHL. It is easy to verify that the HL proposed in [2] is PC. Furthermore, PC-HLHL is an HLHL. It follows that in this graph family, the optimal PC-HLHL size is $\Omega(\sqrt{n})$ times larger than the label size of PC-LHL.

Note that the graph in Figure 3 without the b_i nodes contains non-unique shortest paths from c_i to s. However, one can assign each $\{c_i, p_{i1}\}$ and each $\{p_{i1}, s\}$ edge with a cost of $2/3$, and all other edges with a cost of one. Then the graph has unique shortest paths and the optimal label sizes of the discussed labelings are still the same.

We now modify their example and show the gap between the optimal label size of HHL and the optimal label size of (PC-)HLHL is $\Omega(\sqrt{n})$.

▶ **Theorem 8.** *There is a graph family for which the optimal HHL size is $\Omega(\sqrt{n})$ times larger than the optimal label size of (PC-)HLHL.*

Proof. Consider the undirected graph in Figure 3. The graph consists of k stars, denoted as $c_1, \ldots, c_k$. Each star has k leaves, denoted as $p_{i1}, \ldots, p_{ik}$ for a star with center c_i. The centers of the stars form a clique. The leaves of each star connect to another center denoted as $b_1, \ldots, b_k$. Finally, all leaves connect to the node s. The total number of nodes is $k^2 + 2k + 1$, and the length of every edge is 1.

Now consider the following HLHL for this graph. The nodes are sorted according to their ranking from high to low: $s, b_k, \ldots, b_1, c_k, \ldots, c_1, p_{kk}, \ldots, p_{11}$. The node s is in every label set. The label b_i is in the label set of nodes $p_{i1}, \ldots, p_{ik}$, $c_1, \ldots, c_k$, and b_i. And c_i is in the label set of node c_j where $j \leq i$. Finally, every p node adds itself to its label set. It is easy to verify the cover property and path consistency holds. Each of node s, b and p has a label size of $\mathcal{O}(1)$. The label size of every c node is in $\mathcal{O}(k)$. It follows that the total label size is $\mathcal{O}(n)$.

Now we show the optimal HHL size is in $\Omega(n^{\frac{3}{2}})$. W.l.o.g., let $r(c_1) < r(c_2) < \cdots < r(c_k)$. Consider any p_{ix} and c_j where $i < j$, the shortest path between them is $\pi(p_{ix}, c_j) = (p_{ix}, c_i, c_j)$. Since c_i is not in $L(c_j)$, then either p_{ix} in $L(c_j)$ or c_j in $L(p_{ix})$. Due to the same reason as in Theorem 7, the total label size of HHL is $\Omega(n^{\frac{3}{2}})$. Hence, the gap between the optimal (PC-)HLHL and the optimal HHL is $\Omega(\sqrt{n})$. ◀

In Figure 4, the label size bounds between different labels are shown for general graphs. Clearly, the gaps are at least as large as those for trees. LHL has the smallest optimal label size. Between the LHL and HL labels, the optimal HHL is the largest. There exists a graph family such that every optimal hierarchical labeling of LHL and HL is at least a factor of $\Omega(\sqrt{n})$ larger than the non-hierarchical one. The gap between HHL and (PC-)LHL is at least a factor of $\Omega(\sqrt{n})$ on a graph family that allows multiple shortest paths between two nodes. For unique shortest paths, the known gap is at least a factor of $\Omega(\log n)$ on paths.

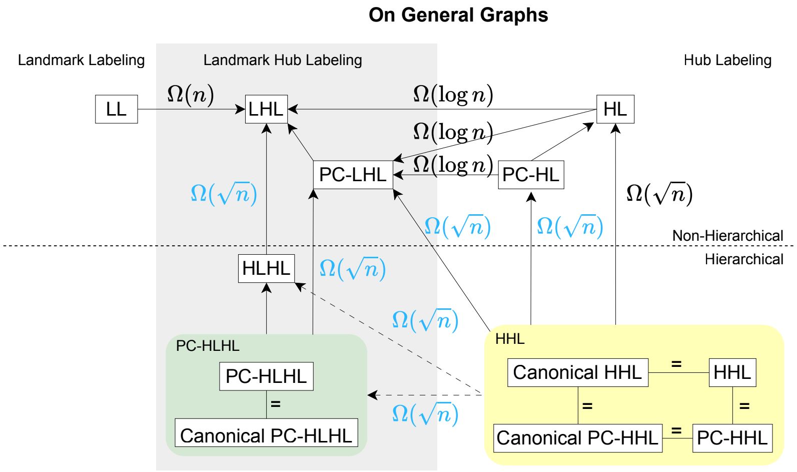


Figure 4 Relation of optimal solution between different labelings. The dashed arrows mean that the gap of the corresponding labelings is found on a graph family with non-unique shortest paths.

4 NP-Completeness of PC-HLHL

In this section, we argue that finding an optimal PC-HLHL is NP-complete by a reduction from HHL. The reduction takes an instance of HHL consisting of a graph $G = (V, E)$ and an integer k and constructs a graph G^+ and an integer k' such that the following conditions are equivalent: (i) There is an HHL of size k in G. (ii) There is an PC-HLHL of size k' in G^+. We prove the following theorem based on this reduction idea.

▶ **Theorem 9.** *The problem of deciding whether an undirected graph has a PC-HLHL of size at most k is NP-complete.*

Before showing the reduction, we first prove the following useful Lemmas. Note that for the cover property, we do not consider the pair of (v, v) for $v \in V$. Therefore, a node v is in its label set if and only if it is in the label set of other nodes.

▶ **Lemma 10.** *Given a graph $G = (V, E)$ in which all nodes in this graph have a degree of at least 3, and there is a unique shortest path between every two nodes. Let $G^+ = (V^+, E^+)$ be the graph obtained by connecting one single node to each node in G. Denote $v' \in V^+ \setminus V$ as the node added to the node $v \in V$, called leaf of v. There is an optimal PC-HLHL such that leaves have lower ranks than non-leaf nodes and no leaf is contained in any label set.*

Proof. Any maximal geodesics that v' lies on contains v too, and v' is always one of the end nodes. Let L be one optimal PC-HLHL on G^+, with the ranking r which is also a canonical PC-HLHL. It was shown in [16] that the canonical PC-HLHL is the minimum PC-HLHL respected to the rank r. We now construct a canonical PC-HLHL ranking r' such that all leaves have a lower rank than its neighbor and the respected labeling L' has a size at most as the size of L. For any $r(u') > r(u)$, let $r'(u) = r(u')$ and $r'(u') = r(u)$, we now show the total label set size does not change. Case one: For a node v such that $u' \in L(v)$, its label set does not increase. Since u lies on all the maximal geodesics that u' lies on. Then $L'(v) \subseteq L(v) \setminus \{u'\} \cup \{u\}$. Case two: For a node v such that $u \notin L(v)$, its label set does not increase either. Assume that after the ranking swap, there is a node v that $u \notin L(v)$, but now u is in $L'(v)$. Then it follows that u has now the highest rank on one maximal geodesic $\pi' \in \pi^+(v, u)$, where $\pi^+(v, u)$ is the set of maximal geodesics that contain both u and v. Let

s, t be the end nodes of the maximal geodesic and $\pi' = \pi(s, v) \cup \pi(v, u) \cup \pi(u, t)$. Then u has now the highest rank on the maximal geodesic $\pi(s, u')$. Then u' has the highest rank on it in the rank r. It follows that u' must be in $L(v)$. It is a contradiction. Therefore, for a node v, if $u' \notin L(v)$, then $u \notin L'(v)$, their label set size does not increase either. We update the rank for such leaves using the same methods until every leaf has a rank lower than their neighbor, and $|L'| = |L|$. It follows that any label set contains no leaf. Furthermore, one can assign r' in the way that all leaves have lower ranks than the nodes in V and it is still optimal. ◀

▶ **Lemma 11.** *Given a graph $G = (V, E)$ and $G^+ = (V^+, E^+)$ as described in Lemma 10. Let L be an optimal PC-HLHL such that all leaves have lower ranks than the nodes in V, then for a leaf v', its label set $L(v') = L(v)$.*

Proof. Let L be an optimal PC-HLHL such that all leaves have lower ranks than the nodes in V. Due to path consistency, for a leaf v', $L(v') \subseteq L(v)$. Now we show $L(v) \subseteq L(v')$. Assume that there is a label $w \in L(v)$ and $w \notin L(v')$. Since $w \notin L(v')$, w does not have the highest rank on any maximal geodesics end by node v', and any shortest path reaches v is a part of one maximal geodesics end by v'. Therefore, on any maximal geodesics that contains v, w does not have the highest rank. According to canonical property, $w \notin L(v)$. This leads to a contradiction. Therefore $L(v) \subseteq L(v')$ and $L(v) = L(v')$. ◀

▶ **Lemma 12.** *Given a graph $G = (V, E)$ and $G^+ = (V^+, E^+)$ as described in Lemma 10. Let L be an optimal PC-HLHL on G^+ such that all leaves have lower ranks than the nodes in V, and $L(G^+[G])$ be the sublabeling of a PC-HLHL in G^+ that is induced by the graph G, then $L(G^+[G])$ is an HHL for nodes in V.*

Proof. Let L be an optimal PC-HLHL such all leaves have lower ranks than non-leaf nodes. We show that there is no landmark in $L(G^+[G])$. Assume that there is a landmark $w \in L(u) \cap L(v) \cap \pi^+(u, v)$, and $L(u) \cap L(v) \cap \pi(u, v) = \emptyset$. It follows $L(u') \cap L(v') \cap \pi(u', v') = \emptyset$. However, L is a valid PC-HLHL. This leads to a contradiction. Therefore, $L(u) \cap L(v) \cap \pi(u, v) \neq \emptyset$ for $u, v \in V$. Hence, $L(G^+[G])$ is an HHL for G. ◀

▶ **Lemma 13.** *Given a graph $G = (V, E)$ and $G^+ = (V^+, E^+)$ as described in Lemma 10. The graph G has an HHL of size k if and only if G^+ has a PC-HLHL of size $2k$.*

Proof. Let $|V| = n$, $|E| = m$. It follows $|V^+| = 2n$ and $|E^+| = m + n$. Assume G has a canonical HHL L of size k with the rank r. We construct a canonical PC-HLHL of G^+ as follows. We assign each non-leaf node v in G^+ with the rank of $r(v) + n$. And assign the rank of leaves randomly from 1 to n. The label set of non-leaf stays the same, and for each leaf v' let $L(v') = L(v)$. Since all maximal geodesics which end by a leaf v' contain the node v, and all shortest paths reaching node v are covered in the HHL, therefore all maximal geodesics are also covered by assigning $L(v') = L(v)$. And the canonical HHL is path-consistent. Hence, the constructed labeling is a canonical PC-HLHL with a size of $2k$.

Assume that L is an optimal PC-HLHL of G^+ of size $2k$. Let L be an optimal PC-HLHL L such all leaves have lower ranks than the nodes in V. According to Lemma 11 and Lemma 12, $L(G^+[G])$ is an HHL for nodes in V with $|L(G^+[G])| = k$. Assuming that $L(G^+[G])$ is not an optimal HHL, then we can construct a PC-HLHL for G^+ with a smaller HHL, then L is not optimal. It is a contradiction. Therefore, the optimal HHL size is k. ◀

5 Construction Algorithms

In [6], the authors formulate HL as a weighted set cover problem (U, S) so that one can use a greedy algorithm to compute an $\mathcal{O}(\log n)$ approximation of the optimal solution. The universe U is a set of uncovered node pairs, S is the collection of all subsets of uncovered pairs that can be covered by the same node. They introduce the center graph of a node v for the undirected graph as follows. In the center graph $G_v = (V', E_v)$, for all $u, w \in V$ there is an edge between node u and w if and only if they are not yet covered and v lies on the shortest path between them. The center graph does not contain isolated nodes. The greedy algorithm chooses the maximal density subgraph (MDS) of all center graphs and adds the center to the label set of the nodes in the MDS. It removes the covered pairs from U and repeats until all pairs are covered. This greedy algorithm can also be applied to LHL [16]. The difference from the HL formulation is that for a center graph G_v, there is an edge between uncovered pair (u, w), if and only if v lies on some shortest path containing u and w. The $\mathcal{O}(\log n)$ approximation factor still holds. We call this algorithm w-LHL.

5.1 Improvements

A naive way to compute the LHL using the greedy algorithm is to precompute all maximal shortest paths (also called geodesics). To construct the center graph of a node v, one must check for each uncovered pair (s, t) whether v lies on one of their maximal geodesics. This can be done by iterating over all maximal geodesics containing v, and checking whether they also contain s and t. However, this takes linear time to check. This implementation costs $\mathcal{O}(n^3)$ space and $\mathcal{O}(n^5)$ running time. We propose two improvements for computing the greedy algorithm efficiently. The improved algorithm needs only $\mathcal{O}(n^2)$ space and has a running time of $\mathcal{O}(n^4 \log n)$. However, it is faster in practice.

The first improvement is to efficiently check whether a node can cover a pair. Instead of storing all maximal geodesics, one can use a distance matrix to store only the shortest path distances. This reduces the space consumption from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$. Given a node v and a pair (s, t), v is a hub or a perfect landmark if and only if $c(s, v) + c(v, t) = c(s, t)$ or $c(s, t) + c(v, t) = c(s, v)$ or $c(s, v) + c(s, t) = c(v, t)$. Therefore, checking whether a node covers a pair can be done in constant time. The second improvement is to avoid unnecessary recomputation of the MDS of a node's center graph every round by using "lazy updates". Note that the MDS of a graph cannot be increased by removing nodes or edges in the graph. We use a max-heap to store for each node the density of the MDS of its center graph. In each iteration, a threshold d is initially set to $-\infty$. While the node v at the top has a corresponding stored density of at least d, we recompute the density md of its new center graph and update $d = \max(d, md)$. Then, the node is pushed back to the heap. Note that the density may decrease because some pairs may be covered from the last round. Finally, we cover the pairs in the best MDS with the respective center node. For the nodes with lower density of the center graph, the MDS is not updated every round to avoid the redundant calculation. The total running time is now in $\mathcal{O}(n^4 \log n)$.

5.2 PC-Labeling

The resulting label from the greedy algorithm is not necessarily PC. Given a valid LHL, one can produce a PC-LHL in polynomial time. We run Dijkstra for each node v until it reaches all nodes w with $v \in L(w)$. Then we check for the resulting shortest path tree whether there is a node u such that $v \notin L(u)$. If so, we add v into the label set of u, thus ensuring PC.

5.3 Hierarchical LHL

In [2], authors propose the weighted greedy HHL algorithm, which selects the whole center graph of maximum density instead of its MDS. Since every center graph is only chosen once, the resulting labeling is hierarchical. We now also select the whole center graph for hierarchical LHL. This algorithm is called w-HLHL. Another greedy algorithm is selecting the center graph which has the most edges. We call this g-(H)LHL. Note that w-HLHL and g-(H)LHL are not necessarily PC. For w-HLHL and g-(H)LHL, one can also apply the "lazy updates". The heap now stores the density of the whole center graph for w-HLHL or the number of edges for g-(H)LHL. The respective running times are in $\mathcal{O}(n^3 \log n)$.

6 Improved Query Answering

In an LHL distance query for nodes $s, t \in V$, we first compute the lower and the upper distance bound LB and UB and then check whether LB is tight by traversing the path from (w.l.o.g.) s to the assumed perfect landmark node v up to distance LB. If we detect t, a path from s to t with cost equal to LB is found and thus this is the correct distance value. The path traversal from s to v can easily dominate the query time.

To improve the query time, we need a more efficient way to characterize whether v is indeed a perfect landmark for s, t. To accomplish this, we now assume that the shortest path tree T_v emerging from v is known. We furthermore denote with $LCA_v(s,t)$ the lowest common ancestor of s and t in the shortest path tree rooted at node v. The following lemma describes the connection between T_v and the role of v for s, t in an LHL query.

▶ **Lemma 14.** *The node v is a perfect landmark for $s, t \in V$ if and only if $LCA_v(s,t) = s$ or $LCA_v(s,t) = t$. The node v is a hub for $s, t \in V$ only if $LCA_v(s,t) = v$.*

Proof. To be a perfect landmark, there needs to be a shortest path from w.l.o.g. s to v over t. Thus, in T_v, t has to be an ancestor of s and $\text{LCA}_v(s,t) = t$ follows. To be a hub, the shortest path from s to t needs to go over v and thus it follows $\text{LCA}_v(s,t) = v$. These observations hold because we have a unique shortest path between each pair of nodes. ◀

It was shown in [5] that for a given tree graph T_v, an LCA data structure can be computed in linear space and time that allows to answer $\text{LCA}_v(s,t)$ queries in $\mathcal{O}(1)$. Below, we discuss how to leverage the lemma and this data structure to improve LHL query times without increasing the asymptotic space consumption and to efficiently check whether a given labeling fulfills the weak cover property.

6.1 An Alternative Method for Label Storage

Conventionally, each node simply stores its labels as a list of nodes with corresponding distance values (sorted by node ID to enable efficient computation of label intersections). But to make use of Lemma 14, we need a label-centered perspective. We thus propose the following alternative way to store label information: Each node $v \in V$ stores a partial shortest path tree T_v^* defined as the smallest subtree of T_v that contains all nodes w with $v \in L(w)$. Shortest path distances are stored along with the tree nodes as well as a pointer to the root node. For each T_v^*, we compute an LCA data structure as described above.

The nodes then additionally hold a list of pointers to their positions in these partial shortest path trees, sorted by the ID of the respective root node. On query time, the lists of s and t are processed and common root node IDs are identified in linear time. For each such root node v, we then query the LCA data structure to check whether v is a perfect landmark for s, t according to Lemma 14. If that is the case, we can immediately abort the procedure and return the respective distance. Otherwise, we check whether $LCA_v(s,t) = v$ and if that is the case, update the UB value (initially set to ∞) if necessary. If no perfect landmark was detected, we return the UB value at the end which is now certified to be correct. We thus get the following corollary.

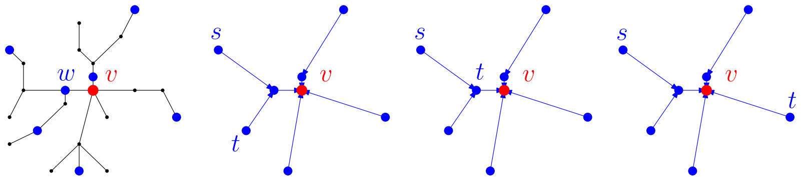


Figure 5 In the left image, the shortest path tree from node v (marked red) is depicted and nodes w with $v \in L(w)$ are marked blue. In the three images to its right, the compressed tree is shown and different query answering scenarios are indicated. In the first, $LCA_v(s,t)$ is neither s, t nor v and thus v is not a perfect landmark or a hub for s, t. In the second, $LCA_v(s,t) = t$ and thus v is a perfect landmark. In the third, $LCA_v(s,t) = v$ and thus v is a potential hub for s, t.

▶ **Corollary 15.** *Distance queries can be answered with LHL in $\mathcal{O}(L_{\max})$.*

However, compared to simply storing $L(v)$ with every node $v \in V$, we might have increased the space consumption significantly by also storing T_v^*. For PC-LHL, this is luckily not the case as shown in the next lemma.

▶ **Lemma 16.** *In a PC-LHL, we have $\sum_{v\in V} |L(v)| = \sum_{v \in V} |T_v^*|$.*

Proof. The PC property demands that for a node $v \in L(w)$ also all nodes u on the shortest path from w to v have v in their label $L(u)$. Therefore, if we consider T_v^*, we know that for each $u \in T_v^*$ we also have $v \in L(u)$. The lemma follows. ◀

Accordingly, the space consumption is at most doubled by the alternative method to store the label information. But as $L(v)$ is replaced by a list of pointers, it demands at most half of the space than it takes to store the node ID and the distance explicitly, reducing the overhead further. Moreover, with T_v^* being stored, we can also answer path queries very efficiently by simply traversing the respective tree edges. To answer path queries with (H)HL efficiently, either substantially larger query times are necessary or additional information needs to be stored with every label [16]. With our method we can guarantee to answer path queries in $\mathcal{O}(L_{\max} + k)$ where k denotes the number of edges on the shortest path.

For non-PC LHL, the summed size of the partial shortest path trees T_v^* might indeed be much larger than the summed label sizes. However, we can use the same trick as proposed in [9] (there applied to the labels themselves for later compression) to also ensure $\sum_{v\in V} |L(v)| = \sum_{v\in V} |T_v^*|$ as for PC-LHL. The idea is simply to only keep nodes $w \in T_v^*$ with $v \in L(w)$ and insert shortcuts between $w, w' \in T_v^*$ if the shortest path $w, .., w', ..v$ is part of T_v but for all nodes u on the shortest path from w to w' it yields $v \notin L(u)$. Figure 5 illustrates this concept as well as how to use the compressed T_v^* for query answering.

So while for the query answering routine described in [16] the PC-property was vital to achieve a query time in $\mathcal{O}(L_{\max} + D)$, we can now guarantee query times in $\mathcal{O}(L_{\max})$ even for non-PC LHL based on our novel method to store the label information.

6.2 LHL Verification

With our improved query answering routine described above, we can also efficiently check whether a given labeling L is a feasible LHL. We compute the shortest path trees rooted at v for all $v \in V$ and the respective LCA data structures. For each node pair s, t, we can then check in T_s the correct distance and compare it to the distance obtained by running the LHL query algorithm. If these do not match for some pair, L is not a valid LHL. This check procedure requires $\mathcal{O}(n^2)$ space, a preprocessing phase in $\mathcal{O}(nm + n^2 \log n)$ and a running time for the verification in $\mathcal{O}(n^2 L_{max})$.

Table 1 Benchmark set summary.

Benchmark Set	# Instances	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert E\rvert/\lvert V\rvert$
OSM	12	[100, 2000]	[99, 2132]	1
PACE	100	[10, 491]	[15, 4100]	[1, 32]

7 Experiments

We implemented the approximation algorithms described in Section 5 in C++, including the lazy variants. The primary goal of the evaluation is to compare HL, HHL, (PC-)LHL and HLHL label sizes in practice. As shown in Table 1, we use road networks extracted from OpenStreetMap[1] (OSM) as well as the PACE challenge 2020 benchmark [2]. The PACE challenge 2020 benchmark contains a diverse set of graphs (100 instances in total) with number of nodes up to 500 nodes and density up to 32. Due to the large running times of the approximation algorithms, we restricted our tests to road networks with up to 2000 nodes (12 graphs in total). Experiments were conducted on a single core of an AMD Ryzen Threadripper 3970X (3.6 GHz) with 128 GB main memory. The time-out was set to 20 minutes per instance.

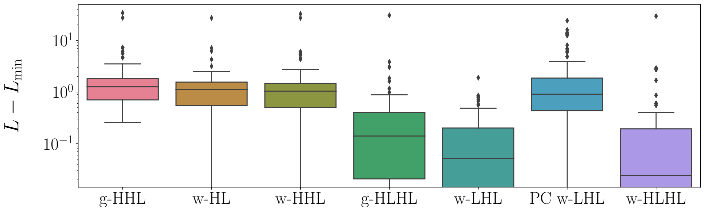


Figure 6 Label size comparison on the PACE benchmark set. The depicted value is the difference between the average label size and the minimal average label size among the solutions of all algorithms. Note the logscale of the y-axis.

Figure 6 and Table 2 show the label size comparison on the PACE benchmark set. In compliance with our theoretical results, the LHL sizes are in general significantly smaller than the HL sizes. In 95 out of 100 instances, the smallest label size is returned from one of the (H)LHL algorithms. The relative difference of HL to LHL size is up to a factor of 34.01 (on a graph with a density of 15). For instances with a density less than 2, w-HLHL computes on average smaller label sizes than w-LHL, which is somewhat surprising. For instances with higher density, w-LHL performs best overall. The PC-LHL constructed with our augmentation algorithm is smaller than w-HL in 51 out of 100 instances but we still observe that enforcing this property increases the label size significantly compared to the other LHL variants. Thus, our new query answering routine described in Section 6, which does no longer rely on the PC-property, is very useful to also keep the label size small.

Figure 7 and Table 3 show the results on the OSM benchmark set. We observe similar trends as for the PACE instances. The LHL algorithms consistently produce smaller label sizes. In fact, w-HLHL computes the smallest labeling in all instances. On average, each

[1] https://www.openstreetmap.org

[2] https://pacechallenge.org/2020/td

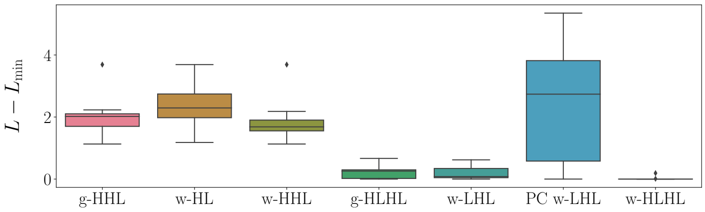


Figure 7 Label size comparison on the OSM benchmark set. The depicted value is the difference between the average label size and the minimal label size among the solutions of all algorithms.

node has at least two more labels in HLs than in LHLs. While this does not sound like a lot, we remark that with every decrease of the average label size by 1 we save space proportional to the size of the graph. The relative difference is up to a factor of 3, even on these rather small instances. We expect this gap to grow further on larger instances. Again, enforcing the PC property leads to larger label sizes and thus should be avoided. Alternatively, there might be other construction algorithms that take the PC property directly into account and thus produce smaller PC-LHL.

Regarding the construction time, the w-HLHL algorithm without lazy updates could not compute the result within 20 minutes on the four largest road network instances, while the engineered one finished in time on all of them. On the large PACE instances, the lazy variant was up to a factor of 3.75 faster.

8 Conclusions & Future Work

We provided new structural insights into LHL, including label size gaps towards the respective HL variants, hardness results, and improved query routines. There are several directions for future work. Our experimental results indicate that w-HLHL performs better than w-LHL on graphs of low density. It would be interesting to provide a theoretical explanation for this. From a practical perspective, there is a clear demand for LHL construction algorithms that scale well with the graph size. Our focus was on approximation algorithms to have a somewhat fair comparison between label sizes. However, most algorithms used in practice are fast heuristics. For LHL, so far only heuristics for the path-consistent variant were investigated [17]. With our alternative method to store labels, there is no longer the need to ensure the PC property, which should allow for easier construction and smaller label sizes. Moreover, there is room for further improving the space consumption. The method for compressing HL described in [9] relies on storing labels as trees and reducing space by identifying common tree structures. This approach should be transferable to LHL, where we store inverse label trees instead. Furthermore, our LCA-based query routine allows for stopping early if a perfect landmark is identified for the query node pair s, t. Thus, assigning node IDs in an LHL such that perfect landmarks are identified early for many node pairs could decrease the query time in practice. This is not possible in an HL query, where always all labels in $L(s)$ and $L(t)$ need to be inspected to ensure that the tightest upper distance bound is identified.

References

1 Haris Angelidakis, Yury Makarychev, and Vsevolod Oparin. Algorithmic and hardness results for the hub labeling problem. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1442–1461. SIAM, 2017.

2 Maxim Babenko, Andrew V Goldberg, Haim Kaplan, Ruslan Savchenko, and Mathias Weller. On the complexity of hub labeling. In *International Symposium on Mathematical Foundations of Computer Science*, pages 62–74. Springer, 2015.

3 Jaime Barceló, Hanna Grzybowska, and Sara Pardo. Vehicle routing and scheduling models, simulation and city logistics. *Dynamic Fleet Management: Concepts, Systems, Algorithms & Case Studies*, pages 163–195, 2007.

4 Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F Werneck. Route planning in transportation networks. *Algorithm engineering: Selected results and surveys*, pages 19–80, 2016.

5 Michael A Bender, Martin Farach-Colton, Giridhar Pemmasani, Steven Skiena, and Pavel Sumazin. Lowest common ancestors in trees and directed acyclic graphs. *Journal of Algorithms*, 57(2):75–94, 2005.

6 Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. Reachability and distance queries via 2-hop labels. *SIAM Journal on Computing*, 32(5):1338–1355, 2003.

7 Daniel Delling, Andrew V Goldberg, Thomas Pajor, and Renato F Werneck. Robust exact distance queries on massive networks. *Microsoft Research, USA, Tech. Rep*, 2, 2014.

8 Daniel Delling, Andrew V Goldberg, Ruslan Savchenko, and Renato F Werneck. Hub labels: Theory and practice. In *Experimental Algorithms: 13th International Symposium, SEA 2014, Copenhagen, Denmark, June 29–July 1, 2014. Proceedings 13*, pages 259–270. Springer, 2014.

9 Daniel Delling, Andrew V Goldberg, and Renato F Werneck. Hub label compression. In *International symposium on experimental algorithms*, pages 18–29. Springer, 2013.

10 PC Gilmore. Families of sets with faithful graph representation. *IBM Research Note NC*, 184, 1962.

11 András Gyárfás and Jenö Lehel. A helly-type problem in trees. *Combinatorial Theory and its applications*, 4:571–584, 1970.

12 Kartik Lakhotia, Qing Dong, Rajgopal Kannan, and Viktor Prasanna. Planting trees for scalable and efficient canonical hub labeling. *arXiv preprint arXiv:1907.00140*, 2019.

13 Sara Nazari, M Reza Meybodi, M Ali Salehigh, and Sara Taghipour. An advanced algorithm for finding shortest path in car navigation system. In *2008 First International Conference on Intelligent Networks and Intelligent Systems*, pages 671–674. IEEE, 2008.

14 Kali Prasad Nepal and Dongjoo Park. Solving the median shortest path problem in the planning and design of urban transportation networks using a vector labeling algorithm. *Transportation Planning and Technology*, 28(2):113–133, 2005.

15 Sven Peyer, Dieter Rautenbach, and Jens Vygen. A generalization of dijkstra's shortest path algorithm with applications to vlsi routing. *Journal of Discrete Algorithms*, 7(4):377–390, 2009.

16 Sabine Storandt. Algorithms for landmark hub labeling. In *33rd International Symposium on Algorithms and Computation (ISAAC 2022)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

17 Sabine Storandt. Scalable landmark hub labeling for optimal and bounded suboptimal pathfinding. In Kate Larson, editor, *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24*, pages 6788–6795. International Joint Conferences on Artificial Intelligence Organization, August 2024. Main Track. `doi:10.24963/ijcai.2024/750`.

A Detailed Experimental Data

Table 2 This table presents a random sample of 30 results, chosen from a total of 100, which describe the average label sizes $|L|$ and running times t(s) of the output produced by each algorithm tested on the PACE challenge benchmark set. The largest label size among them is indicated in italic font, while the smallest label size among them, which is denoted as $L_{\min}$ in Figure 6, is presented in bold font. In column w-LHL, the values of PC represent the number of labels that have increased when the results of w-LHL are transformed into PC w-LHL.

	g-HHL		**w-HL**		**w-HHL**		**g-HLHL**		**w-LHL**			**w-HLHL**	
	$\|L\|$	t(s)	$\|L\|$	t(s)	$\|L\|$	t(s)	$\|L\|$	t(s)	$\|L\|$	t(s)	PC	$\|L\|$	t(s)
e003	*4.9*	≤ 1	*4.9*	≤ 1	*4.9*	≤ 1	**4.1**	≤ 1	**4.1**	≤ 1	0.2	**4.1**	≤ 1
e017	*5.5*	≤ 1	5	≤ 1	5.3	≤ 1	4.8	≤ 1	4.9	≤ 1	0.8	**4.8**	≤ 1
e027	*5.4*	≤ 1	4.8	≤ 1	**4.7**	≤ 1	5	≤ 1	**4.7**	≤ 1	0.7	**4.7**	≤ 1
e029	*5.9*	≤ 1	5.8	≤ 1	5.8	≤ 1	5	≤ 1	**4.9**	≤ 1	0.1	**4.9**	≤ 1
e031	*4.6*	≤ 1	4.5	≤ 1	4.4	≤ 1	4.2	≤ 1	4.1	≤ 1	0.3	**4**	≤ 1
e033	*5.6*	≤ 1	5.4	≤ 1	5.1	≤ 1	5.4	≤ 1	**4.9**	≤ 1	0.3	5.2	≤ 1
e043	*6.8*	≤ 1	6.2	≤ 1	6.6	≤ 1	5.9	≤ 1	**5.6**	≤ 1	0.6	5.8	≤ 1
e047	*9.7*	≤ 1	8.4	≤ 1	8.8	≤ 1	7.7	≤ 1	**7**	≤ 1	0.9	7.2	≤ 1
e059	*14.8*	≤ 1	11.9	≤ 1	12.9	≤ 1	8.3	≤ 1	**7.6**	≤ 1	0.1	8	≤ 1
e065	*6.2*	≤ 1	5.9	≤ 1	5.8	≤ 1	5.6	≤ 1	5.4	≤ 1	0.8	**5.3**	≤ 1
e071	*10.2*	≤ 1	6.9	≤ 1	7.2	≤ 1	8.5	≤ 1	**6.7**	≤ 1	0.4	7.2	≤ 1
e073	*7.8*	≤ 1	7.2	≤ 1	7.5	≤ 1	7.3	≤ 1	**6.7**	≤ 1	1	6.9	≤ 1
e075	*11.4*	≤ 1	9.8	≤ 1	10.9	≤ 1	9.4	≤ 1	**8.8**	≤ 1	2.1	9.1	≤ 1
e083	*10.1*	≤ 1	9.1	≤ 1	9.1	≤ 1	8.8	≤ 1	**8.3**	≤ 1	1.8	8.6	≤ 1
e085	*8.4*	≤ 1	8.3	≤ 1	8	≤ 1	6.9	≤ 1	7.2	≤ 1	1.4	**6.9**	≤ 1
e087	5.8	≤ 1	*6*	≤ 1	5.8	≤ 1	5.9	≤ 1	**5.5**	≤ 1	0.2	5.9	≤ 1
e089	*5.4*	≤ 1	5.3	≤ 1	5.3	≤ 1	5.1	≤ 1	**4.8**	≤ 1	0.2	5.1	≤ 1
e111	*9.2*	≤ 1	8.1	≤ 1	8.5	≤ 1	7.5	≤ 1	**6.8**	1	0.6	7	≤ 1
e117	*9.9*	≤ 1	9.4	1	9.4	≤ 1	9.3	≤ 1	**9**	1	1.8	**9**	≤ 1
e133	*7.5*	1	*7.5*	1	7.4	≤ 1	6.7	≤ 1	**6.4**	1	0.7	6.6	≤ 1
e137	*33.8*	1	33.6	2	33.7	1	7.1	1	**6.4**	2	0.3	6.7	1
e143	8.4	1	*8.6*	1	8.2	≤ 1	**7.6**	≤ 1	**7.6**	1	0.9	**7.6**	≤ 1
e151	5	≤ 1	*5.5*	1	5	≤ 1	4.5	≤ 1	**4.3**	1	0.2	4.5	≤ 1
e153	7.9	1	*8.1*	1	7.9	≤ 1	7.1	1	**6.9**	2	0.9	**6.9**	1
e159	*8.5*	1	*8.5*	1	8.2	1	**7.3**	1	7.5	2	0.7	**7.3**	1
e173	8.5	2	*9.1*	2	8.4	1	7.2	1	7.4	3	1.1	**7.1**	1
e183	*54.9*	19	21.2	8	53.4	2	51.4	6	**20.8**	12	0.3	50.4	6
e185	*9.6*	5	9.5	4	9.5	2	7.9	2	7.9	6	1.1	**7.8**	2
e189	5.3	4	*5.4*	4	5.3	3	4.6	3	4.7	5	0.3	**4.6**	3
e193	*12.9*	26	12.8	13	12.6	6	11.2	7	**10.8**	20	1.2	11.1	7

Table 3 This table presents the average label sizes $|L|$ and running times t(s) of the output produced by each algorithm tested on the benchmark set OSM. The largest label size among them is indicated in italic font, while the smallest label size among them, which is denoted as L_{min} in Figure 7, is presented in bold font. In column w-LHL, the values of PC represent the number of labels that have increased when the results of w-LHL are transformed into PC w-LHL.

	g-HHL		**w-HL**		**w-HHL**		**g-HLHL**		**w-LHL**			**w-HLHL**	
	$\|L\|$	t(s)	$\|L\|$	t(s)	$\|L\|$	t(s)	$\|L\|$	t(s)	$\|L\|$	t(s)	PC	$\|L\|$	t(s)
1	*5.7*	≤ 1	*5.7*	≤ 1	*5.7*	≤ 1	**2.0**	≤ 1	**2.0**	1	0	**2.0**	≤ 1
2	4.7	≤ 1	*4.8*	≤ 1	4.7	≤ 1	3.6	≤ 1	3.6	≤ 1	0.3	**3.5**	≤ 1
3	5.3	≤ 1	*5.5*	≤ 1	5.4	≤ 1	**4.0**	≤ 1	**4.0**	1	0.7	4.4	≤ 1
4	*5.2*	≤ 1	5.1	≤ 1	5.1	≤ 1	**2.9**	≤ 1	3.0	1	0.3	3.0	≤ 1
5	9.5	72	*9.9*	136	9.4	68	8.1	84	8.5	285	4.7	**7.9**	82
6	10.3	74	*10.4*	148	10.2	69	**8.5**	84	8.9	315	4.2	8.7	84
7	9.4	73	*9.5*	146	9.3	68	7.7	82	7.5	311	2.8	**7.4**	83
8	8.9	74	*9.2*	161	8.7	71	7.1	87	7.1	363	2.4	**6.9**	88
9	12.3	484	*13.6*	848	12.0	437	10.8	557				**10.5**	523
10	12.4	494	*13.2*	899	12.3	445	11.0	587				**10.3**	532
11	11.4	475	*11.9*	856	11.0	438	9.7	541				**9.6**	530
12	10.0	489	*10.4*	1019	10.0	461	8.1	584				**7.8**	572

Indexing Graphs for Shortest Beer Path Queries

David Coudert
Université Côte d'Azur, Inria, I3S, CNRS, Sophia Antipolis, France

Andrea D'Ascenzo[1]
Luiss University, Rome, Italy

Mattia D'Emidio
University of L'Aquila, Italy

Abstract

A *beer graph* is an edge-weighted graph $G = (V, E, \omega)$ with *beer vertices* $B \subseteq V$. A *beer path* between two vertices s and t of a beer graph is a path that connects s and t and visits at least one vertex in B. The *beer distance* between two vertices is the weight of a *shortest beer path*, i.e. a beer path having minimum total weight. A *graph indexing scheme* is a two-phase method that constructs an *index* data structure by a one-time preprocessing of an input graph and then exploits it to compute (or accelerate the computation of) answers to *queries* on structures of the graph dataset. In the last decade, such indexing schemes have been designed to perform, effectively, many relevant types of queries, e.g. on reachability, and have gained significant popularity in essentially all data-intensive application domains where large number of queries have to be routinely answered (e.g. journey planners), since they have been shown, through many experimental studies, to offer extremely low query times at the price of limited preprocessing time and space overheads.

In this paper, we showcase that an indexing scheme, to efficiently execute queries on beer distances or shortest beer paths for pairs of vertices of a beer graph, can be obtained by adapting the highway labeling, a recently introduced indexing method to accelerate the computation of classical shortest paths. We design a preprocessing algorithm to build a WHL index, i.e. a weighted highway labeling of a beer graph, and show how it can be queried to compute beer distances and shortest beer paths. Through extensive experimentation on real networks, we empirically demonstrate its practical effectiveness and superiority, in terms of offered trade-off between preprocessing time, space overhead and query time, with respect to the state-of-the-art.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Shortest paths; Information systems → Information systems applications

Keywords and phrases Graph Algorithms, Indexing Schemes, Beer Distances, Algorithms Engineering

Digital Object Identifier 10.4230/OASIcs.ATMOS.2024.2

Supplementary Material
Software (Source Code): `https://github.com/D-hash/ShortestBeerDistanceQueries` [9]

Funding Work partially supported by: (i) Italian Ministry of University and Research through Project "EXPAND: scalable algorithms for EXPloratory Analyses of heterogeneous and dynamic Networked Data" (PRIN grant n. 2022TS4Y3N), funded by the European Union - Next Generation EU; (ii) Gruppo Nazionale Calcolo Scientifico-Istituto Nazionale di Alta Matematica (GNCS-INdAM); (iii) Project EMERGE, innovation agreement between MiSE, Abruzzo Region, Radiolabs, Elital, Leonardo, Telespazio, University of L'Aquila and Centre of EXcellence EX-Emerge (Grant n. 70/2017); (iv) European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on "Telecommunications of the Future" (PE00000001 - program RESTART), project MoVeOver/SCHEDULE ("Smart interseCtions witH connEcteD and aUtonomous vehicLEs", PE00000001, CUP J33C22002880001); (v) French government, through the UCA^JEDI Investments in the Future project managed by the National Research Agency (ANR, reference number ANR-15-IDEX-01.)

[1] Corresponding author.

24th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2024).
Editors: Paul C. Bouman and Spyros C. Kontogiannis; Article No. 2; pp. 2:1–2:18
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Acknowledgements Part of Andrea D'Ascenzo's work was carried out during his Ph.D. studies at University of L'Aquila.

1 Introduction

Determining detours constitutes a major decision process in our daily lives and in modern computing systems: whenever moving from point A to point B, we might need to deviate from the main itinerary in order to stop to a gas station or to buy a beer not to show up empty-handed to a friend one is going to visit. Similarly, in multi-agent systems, we might be interested in planning paths for agents that traverse one of any location of a certain type, for instance to pick-up a specific tool. Finally, in multi-hop communication networks, it might be useful to route data packets through nodes with certain characteristics, e.g. to achieve some form of resiliency. What is the fastest way to accomplish these goals? To model such decision processes from a computational viewpoint, Bacic et al. [3, 4] introduced the notions of *beer graph* and *beer paths*, and defined corresponding optimization problems. A *beer graph* is a graph $G = (V, E, \omega)$, with weight function $\omega : E \mapsto \mathbb{R}^+$ on the edges, where a set of special vertices $B \subseteq V$, called *beer vertices*, is given. A *beer path*, between two vertices s and t of a beer graph, is any path of G from s to t that visits at least one vertex in B whereas a *shortest beer path* for two vertices s and t is a beer path having minimum total weight (called the *beer distance*), i.e. minimizing the sum of the weights of its edges.

Surprisingly, while determining shortest beer paths (and corresponding distances) is a computational problem that arises naturally in a wide range of modern applications, and notwithstanding the fact that such problem can be seen as a special case of the *generalized shortest path* problem [29, 35], algorithmic issues related to such problem have been only recently formalized and investigated [3, 4, 11, 22, 23]. In more details, although a beer path may be a non-simple path (i.e. it might self-intersect), it can be easily shown that any shortest beer path, for a given pair of vertices s, t, always consists of two shortest paths: one from s to a beer vertex, say w, and one from w to t. In other terms, a beer distance can be always determined by finding the minimum, overall beer vertices $w_i \in B$, of the sums of the shortest path distances from s to w_i and from w_i to t. This characterization provides a baseline algorithm for determining shortest beer paths and beer distances for a pair s, t, namely: grow two shortest path trees by Dijkstra's algorithm [16] rooted, respectively, at s and t, and select the beer vertex that minimizes the sum of the distances from s and to t [3, 4]. Unfortunately, while this strategy is simple and considered efficient, as Dijkstra's algorithm runs in almost linear time in the size of the graph [16], many experimental works in the past two decades have shown how employing Dijkstra's algorithm is impractical in many real-world contexts where either the algorithm has to be executed on an interactive basis or when moderately to massively sized networks have to be handled since, in such contexts, it can take up to seconds to compute even a few shortest paths [1, 25, 32].

Practical limitations of shortest path algorithms have motivated the design of several so-called *graph indexing schemes* for shortest paths, i.e. two-phase methods that: (i) perform an *offline*, one-time *preprocessing* phase on the graph to compute auxiliary data, generally stored in a data structure called *index*; (ii) exploit the index, in an *online* phase and upon *query*, for very fast retrieval of shortest paths for (possibly many) pairs of vertices. Due to the excellent performance in practice, combining extremely low query times to find shortest paths (orders of magnitude smaller than methods without indexing) with limited preprocessing time and space overheads (even in large graphs) [1, 2], these schemes have gained significant popularity and have become the state-of-the-art for shortest paths retrieval in all application domains where large number of queries have to be routinely answered (e.g. journey planners or network analytics software) [5, 6, 13–15, 20]. Moreover, such popularity has inspired the development of similar schemes to support, with comparable effectiveness, many other relevant types of queries on graphs (e.g. on reachability [31], on top-k shortest paths [12], on path counts [33], or on communities [34]).

Indexing schemes for queries on beer distances or shortest beer paths have been only very recently considered in the literature. Specifically, some schemes with theoretical guarantees, either on the space occupancy of the index or on preprocessing and query times, have been designed only for special graph classes [4, 11, 23]. For general graphs, instead, no indexing scheme able to support the computation of beer distances or shortest beer paths faster than the baseline, neither theoretically nor in practice, has been developed. To this end, a straightforward way to index a graph for accelerating queries on beer distances and shortest beer paths could be precomputing and storing, into a matrix, distances (or corresponding shortest paths) from all beer vertices to all other vertices of the graph. This approach, which we call B2ALL, translates into a simple and fast query algorithm which can retrieve, (i) the beer distance in $\mathcal{O}(|B|)$ time, by finding the beer vertex minimizing the sum of distances to the two queried vertices; (ii) a shortest beer path of weight ℓ in $\mathcal{O}(|B|\ell)$ time, by unrolling the path that minimizes the sum of the distances. An alternative to the above method, worth being considered, is adapting one of the many indexing schemes designed for the more general and complex problem of determining generalized shortest paths, i.e. paths having minimum weight among those which traverse at least one vertex for each of a set of vertex categories [17, 24, 27, 29, 30, 35]. To apply such methods to beer distances and shortest beer paths, in fact, it suffices to restrict vertex categories to be a single category that contains all beer vertices. For instance, by applying this restriction to the indexing scheme K-SKY, given in [24], one of the best in terms of offered trade-off between preprocessing cost and query performance for generalized shortest paths, one could obtain an algorithm that takes $\mathcal{O}(|B|^2n)$ ($\mathcal{O}(|B|^2n\ell)$, respectively) time to answer a query on the beer distance (on a shortest beer path, respectively), per vertex pair.

However, while both above strategies are appealing in terms of query performance, significantly better than that of two executions of Dijkstra's, it remains unclear whether they are applicable in real-world data-intensive scenarios, due to their high preprocessing time and space occupancy overheads. In both cases, in fact, the preprocessing step takes $\mathcal{O}(n^2|B|)$ time while the resulting index data structure has size $\Omega(n|B|)$ and $\mathcal{O}(n|B|)$, respectively, for any n-vertex graph. To the best of our knowledge, no experimental study has been concerned with the assessment of the average performance of existing indexing methods to support queries on beer distances or shortest beer paths, for both general graphs and special graph classes.

Our Contribution. In this paper, we move in this direction and advance the state-of-the-art with respect to graph indexing methods for queries on beer distances and shortest beer paths in general beer graphs. In particular, we first showcase that an indexing scheme, to efficiently execute such queries, can be obtained by adapting the highway labeling, a recently introduced indexing method to accelerate the computation of classical shortest paths [20]. We propose a preprocessing algorithm, similar but more intuitive than that in [20] for unweighted graphs, to build a WHL index, i.e. a weighted highway labeling of a beer graph; we adapt the query algorithm of [20] to retrieve beer distances or shortest beer paths by only accessing said index. Differently from [20], once the WHL index is computed, our method is oblivious to the graph, in the sense it does need to access it to answer queries. We prove the correctness and analyze the time and space complexities of our methodology.

Then, through extensive experimentation on real networks, we empirically demonstrate its practical effectiveness and superiority, in terms of offered trade-off between preprocessing time, space overhead and query time, with respect to the state-of-the-art. In particular, our experiments show that our query algorithm answers to queries on beer distances, on average, within microseconds per vertex pair, even for very large networks. This is: (i) orders of magnitude faster than both the baseline and the adaptation of the K-SKY method; (ii) comparable to the B2ALL method. At the same time, on the one hand our preprocessing routine preprocesses even very large beer graphs very quickly (within an hour), which is faster than any known indexing scheme for fast beer distance/shortest beer path query answering; on the other hand, our index is compact in size (few hundreds of MBs even for very large networks), with a space occupancy that is up to orders of magnitude smaller than any known index computed for queries on beer distances and shortest beer paths.

Related Works. Bacic et al. [4] have designed a preprocessing-based method that, for *outerplanar graphs*, computes in $\mathcal{O}(n)$ time an index of size $\mathcal{O}(n)$ that allows to find, for a pair of vertices, the beer distance d in $\mathcal{O}(\alpha(n))$ time, where $\alpha(n)$ is the inverse Ackermann function, and a corresponding shortest beer path in $\mathcal{O}(d)$ time. Similarly, Das et al. [11] introduced a data structure for *interval graphs*, occupying $2n \log n + \mathcal{O}(n) + O(|B| \log n)$ bits of space, which allows to compute the beer distance d in $\mathcal{O}(\log^{\epsilon}(n))$ time, for any constant $\epsilon > 0$ and a corresponding shortest beer path in $\mathcal{O}(\log^{\epsilon}(n) + d)$ time. The same authors also show that, if one restricts the input to be a *proper interval graphs*, worst case running time and space occupancy can be further slightly improved [11]. Finally, on a similar line of research it is worth mentioning: (i) the method of Hanaka et al. [23] which, by computing a suited graph decomposition, achieves optimal query time on *series-parallel graphs* and linear preprocessing time on graphs having *bounded-size triconnected components*; (ii) the work of Gudmundsson and Sha [22] which showed how a graph with bounded treewidth t can be preprocessed in $\mathcal{O}(t^3 n)$ time to guarantee the retrieval of the beer distance d in $\mathcal{O}(t^3 \alpha(n))$ time and of the corresponding shortest beer path in $\mathcal{O}(dt^3 \alpha(n))$ time.

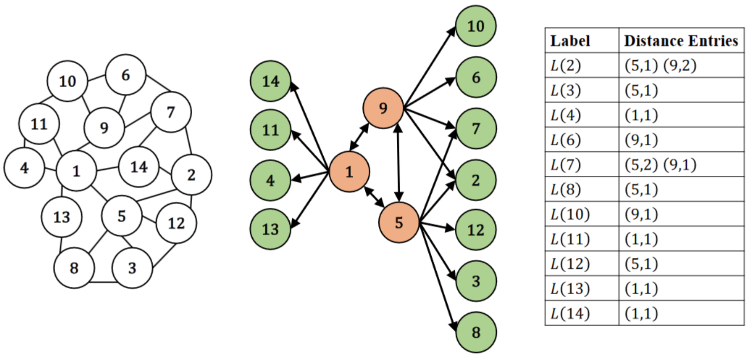
2 Notation and Definitions

We are given a weighted graph $G = (V, E, \omega)$, with $n = |V|$ vertices, $m = |E|$ edges and a *weight function* $\omega : E \mapsto \mathbb{R}^+$ that assigns a positive, real value to each edge of G. A *path* $P = (s = v_1, v_2, \ldots, t = v_\eta)$ in G, connecting a pair of vertices $s, t \in V$, is a sequence of η vertices such that $\{v_i, v_{i+1}\} \in E$ for all $i \in [1, \eta - 1]$. The *weight* $\omega(P)$ of a path P is the sum of the weights of the edges in P. For non-simple paths, path weights include multiplicities of occurrences of a same edge. A *shortest path* P_{st}, for a pair $s, t \in V$, is a path having minimum weight among all those in G that connect s and t. The *distance* $d(s, t)$ between s and t is the weight of a shortest path P_{st}. A *subpath* $(v_i, v_{i+1}, \ldots, v_{j-1}, v_j)$ of a path $P = (v_1, \ldots, v_i, v_{i+1}, \ldots, v_{j-1}, v_j, \ldots, v_\eta)$ is denoted by $P[v_i, v_j]$, for each $0 \leq i < j \leq \eta$. Given a vertex $v \in V$, we denote by $N(v) = \{u \in V | \{v, u\} \in E\}$ the set of neighbors of v. Given a set of vertices $S \subseteq V$ of a graph $G = (V, E, \omega)$, $G[S]$ denotes the subgraph of G induced by S, i.e., $E(G[S]) = \{\{u, v\} \in E \mid u, v \in S\}$. A *beer graph* is a weighted graph $G = (V, E, \omega)$ having a set of *beer vertices* $B \subseteq V$. A *beer path* between two vertices s and t of a beer graph is a path that connects s and t and includes at least one vertex in B. A *shortest beer path* for two vertices s and t is any beer path having minimum weight. The weight of a shortest beer path is called the *beer distance*. In what follows, for the sake of simplicity, we describe our approaches by assuming, w.l.o.g., that the given graph G is connected and undirected. All methods, described in this paper, can be used with digraphs by considering edge orientations and corresponding partitions of the neighbors of a vertex into outgoing and incoming neighbors.

2-Hop-Cover and Highway Labeling. The 2-hop-cover distance labeling is a graph indexing scheme, originally introduced in [7] and heuristically improved in [1]. It is based on the precomputation of an index, called 2-hop-cover distance labeling, that can be used to answer to queries on classical shortest paths and distances, as follows. Given a weighted graph $G = (V, E, \omega)$ and a subset $H \subseteq V$ of its vertices, called *hubs*, a 2-hop-cover distance labeling L of G is a collection of labels $L(v)$, one per vertex $v \in V$ such that: (i) a *label* $L(v)$ is a set of *entries* $(u, d(u, v))$ where $u \in H$; (ii) for each pair of vertices s, t, labels $L(s)$ and $L(t)$ store a set of entries that suffice to compute the distance $d(s, t)$ for any pair of vertices s, t of the graph, that is $d(s, t)$ can be obtained by a function $\delta_L : V \times V \mapsto \mathbb{R}^+$ that takes $L(s)$ and $L(t)$. Note that, it is known that computing a 2-hop-cover distance labeling of minimum size (i.e. number of label entries) is an NP-Hard problem [10].

The highway labeling is an hybrid indexing scheme that generalizes the 2-hop-cover distance labeling, introduced in [20] with the purpose of reducing the preprocessing time and space requirements of the approach of [1], at the price of a slight increase in the average query time. It is considered an hybrid indexing scheme in the sense that queries cannot be solved by accessing only the precomputed data structure and a search of the graph, even if bounded, must be employed to guarantee the correctness of the returned output. Given a graph $G = (V, E, \omega)$, and a subset of its vertices $R \subseteq V$, called *landmarks*, a *highway* $H(R, \delta_H)$ of G with landmarks $R \subseteq V$ is a pair (R, δ_H), where δ_H is a *distance decoding function*, i.e. a function $\delta_H : R \times R \mapsto \mathbb{R}^+$ such that, for any pair $r_i, r_j \in R$, we have $\delta_H(r_i, r_j) = d(r_i, r_j)$. In other words, a highway H is a data structure that stores the distance in G for any two landmarks in R in the form of a function δ_H (e.g. a look-up table). Given a vertex $r \in R \subseteq V$ and two vertices $s, t \in V \setminus R$, an *r-constrained shortest path* P^r_{st} from s to t in G is a path that passes through r (i.e. $r \in P^r_{st}$) and has minimum weight (called *r-constrained distance*) among all paths that connect s and t in G and include r.

Let $H(R, \delta_H)$ be a highway for a graph G with landmarks $R \subseteq V$. A *highway cover distance labeling* (or simply *highway labeling*) of G is a pair $(H(R, \delta_H), L)$ where H is a highway and L is a *labeling* i.e. a collection of labels $L(v)$, one per vertex $v \in V$ such that: (i) a *label* $L(v)$ consists of a set of *entries* in the form $(r_i, d(r_i, v))$ where $r_i \in R$; (ii) for any two vertices $s, t \in V \setminus R$, and for any $r \in R$, labels $L(s)$ and $L(t)$ store entries that suffice to compute the r-constrained distance $d^r(s, t)$. In other terms, any r-constrained distance between two vertices s and t can be found using only the labels of these two vertices. In more details, the latter condition, called *highway cover property* is guaranteed if, for any two vertices $s, t \in V \setminus R$ and for any $r \in R$, there exist $(r_i, d(r_i, s)) \in L(s)$ and $(r_j, d(r_j, t)) \in L(t)$ such that $r_i \in P_{rs}$, for some shortest path P_{rs} from s to t, and $r_j \in P_{rt}$, for some shortest path P_{rt} from r to t, where r_i and r_j may be equal to r. We say $H(R, \delta_H)$ covers G when the highway cover property holds. Moreover, if the label of a vertex v contains an entry $(r, d(r, v))$ for some $r \in R$ we say that vertex v is covered by landmark r in the (highway) labeling. Given any two vertices s and t, a highway labelling L can be used to find any r-constrained distance $d^r(s, t)$ by computing $d(r_i, s) + \delta_H(r_i, r_j) + d(r_j, t)$ where $r_i = r$ or $r_j = r$ and for $(r_i, d(r_i, s)) \in L(s)$ and $(r_j, d(r_j, t)) \in L(t)$. Moreover, an upper bound $d^T(s, t)$ on the shortest path distance from s to t is given by $d^T(s, t) = \min\{d(r_i, s) + \delta_H(r_i, r_j) + d(r_j, t) | (r_i, d(r_i, s)) \in L(s), (r_j, d(r_j, t)) \in L(t)\}$. Observe that such upper bound corresponds to the weight of a shortest path from s to t passing through landmarks r_i and r_j which is exploited by the query routine [20] to compute the true distance $d(s,t)$ by running a distance-bounded bidirectional search on the subgraph $G[V \setminus R]$. In Figure 1 we show an example of highway labeling (H,L) of a graph G, taken from [20]. Consider the graph in the figure (left), the highway H has three landmarks, i.e. $R = \{1,5,9\}$. We have that $\langle 11,1,4\rangle$ is a shortest path between vertices 11 and 4 constrained by landmark 1, i.e. it is 1-constrained shortest path between 11 and 4. In contrast, neither of the paths $\langle 11,10,9,1,4\rangle$ and $\langle 11,4\rangle$ is a 1-constrained shortest path between 11 and 4. In Figure 1 (middle), the outgoing arrows from each landmark point to vertices in G that are covered by this landmark in the highway. The distance labelling in Figure 1 (right) satisfies the highway cover property because for any two vertices that are not landmarks and any landmark $r \in R = \{1,5,9\}$, we can find the r-constrained shortest path distance between these two vertices using their labels and the highway.



Label	Distance Entries
$L(2)$	(5,1) (9,2)
$L(3)$	(5,1)
$L(4)$	(1,1)
$L(6)$	(9,1)
$L(7)$	(5,2) (9,1)
$L(8)$	(5,1)
$L(10)$	(9,1)
$L(11)$	(1,1)
$L(12)$	(5,1)
$L(13)$	(1,1)
$L(14)$	(1,1)

Figure 1 Example of Highway Labeling of a graph [20].

3 Indexing Scheme for Beer Distance Queries

In this section, we describe a graph indexing scheme to support queries on beer distances and shortest beer paths. Specifically, we first introduce an algorithm that preprocesses a beer graph to compute a WHL index, a data structure that can be used through a dedicated query algorithm, also given here, to compute beer distances and shortest beer paths. W.l.o.g, and for the sake of simplicity, we describe our method to support the execution of beer distance (BD, for short) queries only. All given approaches can be easily extended to return a corresponding shortest beer paths by the strategies similar to those used for shortest paths, e.g. by storing predecessors (see, e.g., [10, 14]).

Our methodology is based on the work of Farhan et al. [20]. The key idea is, given a beer graph $G = (V,E,\omega)$ with beer vertices B, to compute a highway labeling $X = (H,L)$ that is; (i) weighted; (ii) considers, as set of landmarks R, the set of beer vertices B; (iii) covers G. In fact, it is easy to show that a highway labeling that satisfies the above three properties can be used to retrieve beer distances for any pair of vertices of the given beer graph by the following observations. Specifically, while using this structure alone does not suffice to compute shortest path distances (see Section 2), it is easy to see that, under the above assumptions, the beer distance, for any pair of vertices $s,t \in V$, is returned in $\mathcal{O}(|B|^2)$ time by the following query routine:

$$Q(s,t,X) = \min_{b_i,b_j \in B}\{d(b_i,s)+\delta_H(b_i,b_j)+d(b_j,t) | (b_i,d(b_i,s)) \in L(s),(b_j,d(b_j,t)) \in L(t)\} \quad (1)$$

Indeed, by the highway cover property, any r-constrained distance $d^r(s,t)$ for landmark $r \in R$ can be found by computing $d(r_i,s) + \delta_H(r_i,r_j) + d(r_j,t)$ where $r_i = r$ or $r_j = r$ and for $(r_i, d(r_i,s)) \in L(s)$ and $(r_j, d(r_j,t)) \in L(t)$. Thus, the weight of a shortest beer path that includes a beer vertex $b \in B$ can be found by determining $d(r_i,s) + \delta_H(r_i,r_j) + d(r_j,t)$ where $r_i = b$ or $r_j = b$ and for $(r_i, d(r_i,s)) \in L(s)$ and $(r_j, d(r_j,t)) \in L(t)$. It follows that the upper bound $d^T(s,t)$ on the shortest path distance from s to t, provided by the labeling as $d^T(s,t) = \min\{d(r_i,s) + \delta_H(r_i,r_j) + d(r_j,t) | (r_i, d(r_i,s)) \in L(s), (r_j, d(r_j,t)) \in L(t)\}$, is the minimum of the weights of all r-constrained shortest paths, for each $r \in B$, which corresponds to the minimum of the weights of the shortest paths that include a beer vertex $b \in B$ for all beer vertices of B, which is the beer distance. We call a weighted highway labeling built on beer vertices, that satisfies the above equivalency, a WHL index.

Now, Farhan et al. [20] have shown that a highway labeling can be found in linear time for *unweighted* graphs by running $|R|$ modified breadth-first search (BFS) visits, one per landmark, and by incrementally constructing labels that satisfy the following. Whenever a vertex v is extracted from the queue of the BFS, rooted at landmark $r_i \in R$, if we call P_{r_iv} the shortest path from r_i to v traversed by the search, and $d(r_i,v)$ its weight, then: (i) if vertex $v \in V \setminus R$, an entry $(r_i, d(r_i,v))$, for some $r_i \in R$, is added to $L(v)$ if and only if there does not exist any other landmark in P_{r_iv}, i.e. $P_{r_iv} \cap R = \{r_i\}$; (ii) if v is a landmark $r \in R \setminus \{r_i\}$, then $d(r_i,v)$ is added to the highway, i.e. to $\delta_H(r_i,v)$ to build the distance decoding function (e.g. in the form of a table). Unfortunately, no strategy for efficiently building a weighted highway labeling covering a weighted graph is given in [20]. In subsequent works, namely [18, 19], the authors claim that the construction of [20] can be adapted to weighted graphs by replacing the modified BFS with Dijkstra's algorithm. Differently from an unweighted one, in fact, both the label entries of a weighted highway labeling and the associated distance decoding function must store weights of shortest paths (i.e. sums of weights of edges), rather than path lengths (i.e. number of edges). However, no details on the extension are given in [18,19] and adapting the approach from unweighted to weighted inputs seems to be all but straightforward. More specifically, in the unweighted version, each visit, rooted at a given vertex r_i, employs two queues, called $\mathcal{Q}_{label}$ and $\mathcal{Q}_{prune}$. The former is used for paths to vertices that do not traverse landmarks other than r_i, while the latter for paths that traverse at least one landmarks $r_j, j \in R \setminus \{r_i\}$. When an element is extracted from $\mathcal{Q}_{label}$ with some priority δ, all neighbors of the terminal vertex of the path are enqueued , with priority $\delta + 1$ into $\mathcal{Q}_{label}$ (if they are not landmarks) or into $\mathcal{Q}_{prune}$ (otherwise). In the latter case, elements in $\mathcal{Q}_{prune}$ having priority δ are all extracted, and neighboring vertices are enqueued in the same queue with priority $\delta + 1$. If the graph is unweighted, this mechanism guarantees that a vertex is never added to both queues. However, it is easy to see that this would not be true if the treated input graph is weighted, with simultaneous occurrences of a same vertex in both queues leading to unnecessary queue operations that have to be managed by the algorithm. Therefore, in what follows, we introduce a novel algorithm, called BuildWhl, to efficiently compute a WHL index, i.e. a weighted highway labeling with beer vertices as landmarks that covers a beer graph, using only one priority-queue.

Algorithm BuildWhl works as follows. Starting from each beer vertex in B as root, we run a modified version of Dijkstra's algorithm that searches the graph by relying on a single one-level priority queue (e.g. a min-heap). For each root $r \in B$, the algorithm assigns, to each vertex $v \in V$: a flag $flag[v]$ initially `false`; a tentative distance $d[v]$ initially equal to some infinity default value. Then, the visit starts by enqueueing the root with zero priority and, whenever a vertex v is dequeued, say with associated priority δ, two cases can occur. If v is a beer vertex $v \neq r$, first its flag is set to `true`, to trace the fact that a path from the root to such vertex traverses a beer vertex. Then, the associated value δ, corresponding to the weight $d(r, v)$ of a shortest path from r to v, is stored in the highway H (i.e. the distance decoding function is built). Moreover, regardless of whether v is a beer vertex or not, if the value of its flag is `true`, then value δ, which equals distance $d(r, v)$, is discarded (since the shortest path inducing such distance traverses a beer vertex). Viceversa, if the flag of v is `false`, we have that value δ is the distance from r to v induced by a shortest path, say P_{rv}, that does not traverse any beer vertex, thus entry (r, δ) is added to $L(v)$. Finally, a distance relaxation operation is performed on the neighbors of v. In details, for each $w \in N(v)$, we check if $d[w] > \delta + \omega(v, w)$, i.e. if the weight of the path from r to v plus the weight of edge (v, w) is less than the tentative distance $d[w]$. In the affirmative case, it follows that the path P_{rv} to v combined with edge (v, w) has weight that is smaller than any previously discovered path to w. Hence, we update $d[w] = \delta + \omega(v, w)$ and either enqueue w with priority $d[w]$ or decrease its current priority to $d[w]$. Contextually, we update the flag of w to that of v, to keep trace of traversal of beer vertices. Viceversa, the path from r to w through v is either not a shortest path to w or a shortest path of equal weight, hence it is not considered. The pseudo-code of BuildWhl is given in Algorithm 1. We next state

Algorithm 1 Algorithm BuildWhl.

```
Input: A beer graph $G = (V, E, \omega)$ with beer vertices $B \subseteq V$.
Output: A WHL index $(H, L)$.
foreach $r \in R$ do
    foreach $v \in V$ do
        $d[v] \leftarrow \infty$;
        $flag[v] \leftarrow$ false;
    $PQ \leftarrow \emptyset$;                          // $PQ$ is a priority queue, e.g. min-heap
    Enqueue vertex $r$ into $PQ$ with priority 0;
    while $PQ \neq \emptyset$ do
        Dequeue from $PQ$ vertex $v$ having minimum priority $\delta$;      // Here $d[v] = \delta$
        if $v \in B \setminus \{r\}$ then
            $flag[v] \leftarrow$ true;
            $\delta_H(r, v) \leftarrow \delta$;        // Store $\delta$ into entry $\delta_H(r, v)$ of the highway $H$
        if $flag[v]$ is false then
            Add $(r, \delta)$ to $L(v)$;
        foreach $w \in N(v)$ do
            if $d[w] > \delta + \omega(v, w)$ then
                $flag[w] \leftarrow flag[v]$;
                $d[w] \leftarrow \delta + \omega(v, w)$;
                if $d[w] = \infty$ then Enqueue vertex $w$ into $PQ$ with priority $d[w]$;
                else Decrease priority of vertex $w$ in $PQ$ to $d[w]$;
```

the correctness and the running time of procedure BuildWhl. In particular, we are able to prove that algorithm BuildWhl computes a weighted highway labeling that covers the input graph with landmarks B. This implies that the query routine of Eq. 1 on said labeling returns the beer distance for every pair of vertices.

▶ **Lemma 1.** *Algorithm 1 adds entry (r, δ) to label $L(v)$ of a vertex $v \in V \setminus B$ if and only if $r \in B$ is the only beer vertex in the shortest path P_{rv} inducing $\delta = d(r, v)$.*

Proof. Suppose that $P_{rv} \cap B \neq \{r\}$, i.e. there exists at least a beer vertex $r' \in B \setminus \{r\}$ in the shortest path P_{rv} between r and v that causes v to be enqueued (i.e. path having weight $\delta = d(r, v)$). Since r' lies in P_{rv}, it must be extracted from the priority queue before v, and

Line 10 in Algorithm 1 sets the flag of r' to `true`. By the subpaths optimality property of shortest paths, it follows that flag of r' is propagated in each distance relaxation operation of any vertex $v' \in P_{rv}[r', v]$ (cf. Lines 15-16). Therefore, when v is extracted from the priority queue, its flag is equal to `true`, and Line 13 is not executed. On the other hand, if Algorithm 1 inserts (r, δ) in $L(v)$ in Line 13, then we have $P_{rv} \cap B = \{r\}$. In fact, Line 13 is executed only if the flag of vertex v is set to `false`, which happens if and only if each distance relaxation applied on the vertices $v' \in P_{rv}$ was not induced by a path traversing a landmark $r' \neq r$. ◀

By Lemma 1 we can derive a corollary similar to that given in [20] for unweighted graphs.

▶ **Corollary 2.** *Let $r \in B$ be a beer vertex and let $v \in V \setminus B$ be a non-beer vertex. Let (H, L) be a labeling constructed by Algorithm 1. If $(r, d(r, v)) \notin L(v)$, then there must exist a beer vertex r_j such that $(r_j, d(r_j, v)) \in L(v)$, and $d(r, v) = d(r_j, v) + \delta_H(r_j, r)$.*

Finally, we can prove that the labeling computed by BuildWhl covers the given graph.

▶ **Theorem 3.** *The highway labeling (H, L) constructed by Algorithm 1 on a beer graph $G = (V, E, \omega)$ satisfies the highway cover property for G.*

Proof. We need to show that, for any two vertices $s, t \in V \setminus B$ and any $r \in B$, there exist $(r_i, d(r_i, s)) \in L(s)$ and $(r_j, d(r_j, t)) \in L(t)$ such that $r_i \in P_{rs}$ and $r_j \in P_{rt}$. To that aim, we can apply Corollary 2 to the following four cases: (i) r covers both s and t; (ii) r covers s but not t; (iii) t is covered by r, while s is not; (iv) neither s nor t is covered by r. For the sake of completeness, we also give it here. With a slight abuse of notation, in what follows we use $r' \in L(v)$ to denote that landmark r' covers a vertex v, i.e. that there exists an entry $(r', d(r', v))$ in $L(v)$. In Case (i), we have $r \in L(s)$ and $r \in L(t)$, thus $r = r_i = r_j$. Case (ii): $r_i = r$, while Corollary 2 ensures the existence of another landmark r_j such that r_j is in the shortest path between t and r and $r_j \in L(t)$. Case (iii) is treated similarly to the previous case. Finally, again by Corollary 2 applied to Case (iv), we know that there exist two landmarks r_i, r_j such that r_i (r_j, respectively) is in the shortest path between s (t, respectively) and r, and $(r_i, d(r_i, s)) \in L(s)$ and $(r_j, d(r_j, t)) \in L(t)$. ◀

▶ **Theorem 4.** *Algorithm 1 runs in $\mathcal{O}(|B|(m + n \log n))$ time.*

Proof. Observe that, for each landmark $r \in B$, the algorithm performs a call to a Dijkstra-like algorithm for each landmark. During the execution of such routine, rooted at a landmark r, the algorithm update the flags of the vertices. The initialization of the flags takes $\mathcal{O}(n)$ time. Then, when a vertex v is extracted from the min-heap data structure PQ, if $v \in B \setminus \{r\}$, the flag of v is set to `true` and the algorithm inserts distance δ into highway H in constant time. Otherwise, the algorithm adds entry (r, δ) to label $L(v)$, which again can be done in constant time. Finally, the algorithm checks if value $d[w]$ can be decreased for neighbors w of v. In the affirmative case, the flag of w is set to the flag of v and a queue operation is performed, in $\mathcal{O}(\log n)$ time. Overall, the maintenance of the flags requires $\mathcal{O}(m + n)$ constant time operations while queue operations account for $\mathcal{O}(n \log n)$ time. Furthermore, the algorithm performs $|B| - 1$ insertions into the highway H and at most $n - |B|$ insertions into the labels of the vertices, so overall $\mathcal{O}(n)$ insertions operations, each taking $\mathcal{O}(1)$ time. Therefore, the time complexity per landmark $r \in B$ is $\mathcal{O}(m + n \log n)$ and the claim follows. ◀

4 Experimental Evaluation

In this section, we present the results of an experimental evaluation we conducted to assess the effectiveness of our new graph indexing method for BD queries.

Setup and Executed Tests. We implemented: (i) algorithm BuildBM that precomputes and stores, in a corresponding b2all matrix, all distances from beer vertices to other vertices of the graph; (ii) the corresponding query algorithm, called QueryBM, that solves a BD query for a pair of vertices by determining the minimum of the sums of the distances between all beer vertices and the two queried vertices; (iii) our method BuildWhl to compute a whl index; (iv) the corresponding query algorithm, denoted by QueryWhl, that computes the BD for a pair of vertices by accessing the index as per Eq. 1; (v) the baseline method, denoted by BaseLine, which executes Dijkstra's algorithm twice to compute the shortest path trees rooted at the two queried vertices s, t and selects the beer vertex providing the minimum sum of distances from s and to t. All our code is written in C++, compiled with GCC 10.5 with opt. level $O3$. All tests have been executed on a workstation equipped with an Intel Xeon© CPU E5-2643, clocked at 3.40 GHz, and 96 GB of RAM, running Ubuntu Linux.

Input Instances. As inputs to our experiments we considered real-world road graphs taken from publicly available repositories [26]. Details on used inputs, including number of vertices $|V|$ and edges $|E|$, average vertex degree, and size of the graph file $|G|$ are reported in Table 1. Graphs are sorted from top to bottom by $|V| + |E|$. Concerning the number of beer vertices b, to study the scalability properties of the proposed approach, as suggested in [28], we measure and report performance indicators for doubling values of parameter b that are relevant to the domain, i.e. for $b \in \{25, 50, 100, 200, 400\}$. Beer vertices are placed in each graph by a strategy called *distance-δ bounded dominating set*, a policy that is often considered in network design problems, and adopted in real-world scenarios, to identify a subset of important vertices that have to be traversed by paths between other vertices (e.g. routers responsible for specific messages) [21]. For each input graph and value of b, we: (i) run BuildWhl to construct a whl index; (ii) execute BuildBM to precompute the b2all matrix. We measure the running times of both BuildWhl and BuildBM (denoted by PT_{whl} and PT_{b2all}, respectively), and the space occupancy, in MB, of the whl index (which includes both the distance deconding function and the label entries) and the b2all matrix (denoted by IS_{whl} and IS_{b2all}, respectively). Note that, in all our experiments, we represent distances and landmarks/beer vertices as 32-bit integers. Moreover, we execute QueryWhl and QueryBM to solve 10^5 BD queries for randomly selected vertex pairs and measure their average execution time (denoted by QT_{whl} and QT_{b2all}, respectively). Finally, we run algorithm BaseLine for a subset of 10^4 of the aforementioned vertex pairs, and measure the average execution time. Such reduction is necessary due to the moderately large running time per query of BaseLine.

Table 1 Overview of Input Graphs.

Graph	$\|V\|$	$\|E\|$	Avg. Deg.	$\|G\|$
lux	30 647	37 773	2.46	1.1 MB
ny	264 346	365 050	2.76	13 MB
bay	321 270	397 415	2.47	14 MB
col	435 666	521 200	2.39	19 MB
dnk	469 110	545 019	2.32	18 MB
fla	1 070,376	1 343 951	2.51	48 MB
nw	1 207 945	1 410 387	2.33	52 MB
ne	1 524 453	1 934 010	2.53	71 MB
cal	1 890 815	2 315 222	2.44	87 MB
ita	2 077 709	2 589 431	2.49	91 MB
deu	4 047 577	4 907 447	2.42	178 MB
usa	23 947 347	28 854 312	2.40	1.2 GB

Analysis. The results of our experimentation are summarized in Tables 2–4. For both performance indicators indexing time and space occupancy, in Tables 2–4 we give the ratio of the value achieved by B2ALL and that achieved by WHL. In terms of indexing/preprocessing time, our data show that algorithms BUILDWHL and BUILDBM are comparable and both can be considered reasonably practical since both average execution times range from less than half a second in the smallest instance (i.e. LUX with $b = 25$) to around one hour in the largest one (i.e. USA with $b = 400$). However, we notice that BUILDWHL is always faster than BUILDBM, by factors that span in the orders of tens of percentage points (see $\text{PT}_{\text{B2ALL}}/\text{PT}_{\text{WHL}}$ column in Tables 2–4). This is most likely due to the fact that BUILDWHL stores label entries only when the flag of a vertex is false, i.e. when the root r is the only beer vertex lying on the path found by the visit. Furthermore, to characterize of the scalability properties of BUILDWHL, in Fig 2 (left) and 3 (left) we plot its running time as a function of b for all inputs. Our data suggest a linear trend of the indexing time with respect to to b, which matches our analysis of Theorem 4. On top of that, Figures 2 (middle) and 3 (middle) suggest that BUILDWHL scales better with respect to b than BUILDBM, in terms of running time, with the ratio between the execution times of the two algorithms increasing with b.

Table 2 Results of the experimentation with $b = 25$ beer vertices.

Graph	Preprocessing			Space Occupancy			Query (s)		
	PT$_{\text{WHL}}$ (s)	PT$_{\text{B2ALL}}$ (s)	PT$_{\text{B2ALL}}$/PT$_{\text{WHL}}$	IS$_{\text{WHL}}$ (MB)	IS$_{\text{B2ALL}}$ (MB)	IS$_{\text{B2ALL}}$/IS$_{\text{WHL}}$	QT$_{\text{WHL}}$	QT$_{\text{B2ALL}}$	BASELINE
LUX	0.12	0.13	1.09	0.13	2.92	22.46	$1.2 \cdot 10^{-7}$	$1.1 \cdot 10^{-7}$	0.01
NY	1.65	1.78	1.08	3.80	25.21	6.63	$4.5 \cdot 10^{-7}$	$3.2 \cdot 10^{-7}$	0.10
BAY	1.94	2.40	1.24	2.59	30.64	11.83	$4.3 \cdot 10^{-7}$	$3.7 \cdot 10^{-7}$	0.13
COL	2.39	2.96	1.24	1.77	41.55	23.47	$3.5 \cdot 10^{-7}$	$3.3 \cdot 10^{-7}$	0.18
DNK	2.56	3.30	1.29	5.36	44.74	8.35	$4.1 \cdot 10^{-7}$	$3.8 \cdot 10^{-7}$	0.17
FLA	6.95	7.40	1.07	12.36	102.08	8.26	$4.4 \cdot 10^{-7}$	$3.4 \cdot 10^{-7}$	0.43
NW	7.25	8.43	1.16	8.30	115.20	13.88	$4.2 \cdot 10^{-7}$	$3.5 \cdot 10^{-7}$	0.51
NE	11.04	12.41	1.12	21.63	145.38	6.72	$4.6 \cdot 10^{-7}$	$3.9 \cdot 10^{-7}$	0.70
CAL	13.63	15.43	1.13	15.44	180.32	11.68	$5.2 \cdot 10^{-7}$	$4.3 \cdot 10^{-7}$	0.87
ITA	10.56	12.62	1.19	7.93	198.15	24.99	$4.9 \cdot 10^{-7}$	$5.1 \cdot 10^{-7}$	0.72
DEU	26.81	31.80	1.19	48.44	386.01	7.97	$6.9 \cdot 10^{-7}$	$5.2 \cdot 10^{-7}$	1.63
USA	246.94	280.49	1.14	91.35	2283.80	25.00	$9.7 \cdot 10^{-7}$	$6.7 \cdot 10^{-7}$	10.90

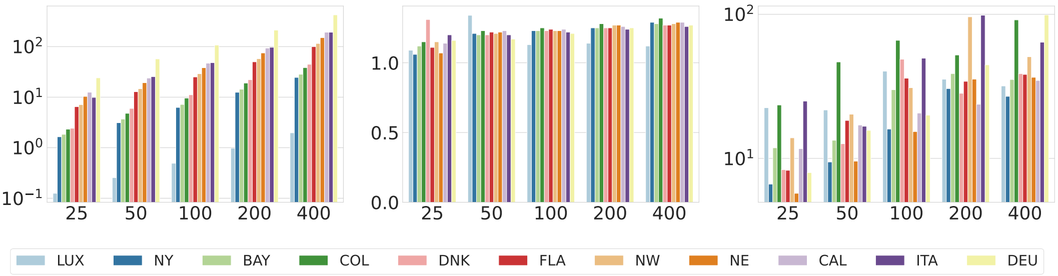


Figure 2 Running time (in seconds) of BUILDWHL (left); ratio of the running time of BUILDWHL to that of BUILDBM (middle); ratio of the space occupancy of the B2ALL matrix index to that of the WHL (right), for all graphs, except USA, as a function of b (x-axis).

Concerning the sizes of the WHL index and the B2ALL matrix, instead, we observe that our new scheme significantly outperforms method B2ALL. In fact, BUILDWHL computes very compact WHL indices, even for large graphs, with an average space occupancy that is up to orders of magnitude smaller than that of B2ALL matrices, which contain precisely one distance value per beer vertex and for all vertices of the graph (cf. column “Space Occupancy” of Tables 2-4). In details, we observe that, even in the largest considered graph, i.e. USA (see

Table 3 Results of the experimentation with $b \in \{50, 100\}$ beer vertices.

b	Graph	Preprocessing			Space Occupancy			Query (s)		
		PT_{WHL} (s)	$\mathrm{PT}_{\text{B2ALL}}$ (s)	$\mathrm{PT}_{\text{B2ALL}}/\mathrm{PT}_{\text{WHL}}$	IS_{WHL} (MB)	$\mathrm{IS}_{\text{B2ALL}}$ (MB)	$\mathrm{IS}_{\text{B2ALL}}/\mathrm{IS}_{\text{WHL}}$	QT_{WHL}	$\mathrm{QT}_{\text{B2ALL}}$	BaseLine
50	LUX	0.25	0.34	1.34	0.27	5.85	21.67	$1.5 \cdot 10^{-7}$	$1.4 \cdot 10^{-7}$	0.01
	NY	3.11	3.77	1.21	5.34	50.42	9.44	$3.5 \cdot 10^{-7}$	$3.4 \cdot 10^{-7}$	0.11
	BAY	3.68	4.41	1.20	4.59	61.28	13.35	$3.3 \cdot 10^{-7}$	$2.2 \cdot 10^{-7}$	0.13
	COL	4.77	5.89	1.23	1.78	83.10	46.69	$4.9 \cdot 10^{-7}$	$2.2 \cdot 10^{-7}$	0.17
	DNK	5.97	7.14	1.20	7.08	89.48	12.64	$3.8 \cdot 10^{-7}$	$2.4 \cdot 10^{-7}$	0.21
	FLA	12.82	15.59	1.22	11.14	204.16	18.33	$4.2 \cdot 10^{-7}$	$2.5 \cdot 10^{-7}$	0.46
	NW	14.67	17.79	1.21	11.37	230.40	20.26	$4.3 \cdot 10^{-7}$	$2.7 \cdot 10^{-7}$	0.55
	NE	19.29	23.46	1.22	30.37	290.77	9.57	$4.4 \cdot 10^{-7}$	$2.6 \cdot 10^{-7}$	0.71
	CAL	23.86	29.27	1.23	21.17	360.64	17.04	$4.5 \cdot 10^{-7}$	$2.7 \cdot 10^{-7}$	0.88
	ITA	25.47	30.61	1.20	23.73	396.29	16.70	$4.6 \cdot 10^{-7}$	$2.9 \cdot 10^{-7}$	0.96
	DEU	57.03	66.62	1.17	49.37	772.01	15.64	$6.0 \cdot 10^{-7}$	$3.5 \cdot 10^{-7}$	2.05
	USA	412.75	469.48	1.13	161.71	4 567.59	28.24	$5.7 \cdot 10^{-7}$	$3.3 \cdot 10^{-7}$	13.91
100	LUX	0.49	0.55	1.13	0.29	11.69	40.31	$1.4 \cdot 10^{-7}$	$1.7 \cdot 10^{-7}$	0.01
	NY	6.25	7.68	1.23	6.31	100.84	15.98	$3.7 \cdot 10^{-7}$	$2.9 \cdot 10^{-7}$	0.11
	BAY	7.16	8.83	1.23	4.09	122.55	29.96	$3.4 \cdot 10^{-7}$	$2.9 \cdot 10^{-7}$	0.13
	COL	9.60	12.03	1.25	2.52	166.19	65.95	$4.1 \cdot 10^{-7}$	$3.0 \cdot 10^{-7}$	0.18
	DNK	11.11	13.71	1.23	3.67	178.95	48.76	$3.8 \cdot 10^{-7}$	$3.0 \cdot 10^{-7}$	0.20
	FLA	25.08	31.12	1.24	11.34	408.32	36.01	$4.3 \cdot 10^{-7}$	$3.4 \cdot 10^{-7}$	0.46
	NW	29.08	35.90	1.23	14.89	460.79	30.95	$4.4 \cdot 10^{-7}$	$3.7 \cdot 10^{-7}$	0.55
	NE	38.17	47.12	1.23	37.89	581.53	15.35	$4.7 \cdot 10^{-7}$	$3.6 \cdot 10^{-7}$	0.71
	CAL	47.01	58.20	1.24	34.96	721.29	20.63	$4.9 \cdot 10^{-7}$	$3.6 \cdot 10^{-7}$	0.88
	ITA	48.14	58.72	1.22	15.98	792.58	49.60	$4.8 \cdot 10^{-7}$	$3.6 \cdot 10^{-7}$	0.92
	DEU	107.30	130.06	1.21	77.34	1 544.03	19.96	$5.2 \cdot 10^{-7}$	$3.8 \cdot 10^{-7}$	1.98
	USA	836.25	981.31	1.17	341.28	9 135.19	26.76	$6.5 \cdot 10^{-7}$	$4.2 \cdot 10^{-7}$	13.81

Table 4 Results of the experimentation with $b \in \{200, 400\}$ beer vertices.

b	Graph	Preprocessing			Space Occupancy			Query (s)		
		PT_{WHL} (s)	$\mathrm{PT}_{\text{B2ALL}}$ (s)	$\mathrm{PT}_{\text{B2ALL}}/\mathrm{PT}_{\text{WHL}}$	IS_{WHL} (MB)	$\mathrm{IS}_{\text{B2ALL}}$ (MB)	$\mathrm{IS}_{\text{B2ALL}}/\mathrm{IS}_{\text{WHL}}$	QT_{WHL}	$\mathrm{QT}_{\text{B2ALL}}$	BaseLine
200	LUX	0.97	1.11	1.14	0.66	23.38	35.42	$1.7 \cdot 10^{-7}$	$2.9 \cdot 10^{-7}$	0.01
	NY	12.46	15.56	1.25	6.61	201.68	30.51	$4.4 \cdot 10^{-7}$	$4.0 \cdot 10^{-7}$	0.11
	BAY	14.33	17.95	1.25	6.32	245.11	38.78	$3.7 \cdot 10^{-7}$	$4.1 \cdot 10^{-7}$	0.13
	COL	19.00	24.28	1.28	6.37	332.39	52.18	$4.1 \cdot 10^{-7}$	$4.2 \cdot 10^{-7}$	0.17
	DNK	22.15	27.77	1.25	12.65	357.90	28.29	$4.2 \cdot 10^{-7}$	$4.3 \cdot 10^{-7}$	0.20
	FLA	49.94	62.58	1.25	23.84	816.63	34.25	$4.9 \cdot 10^{-7}$	$4.5 \cdot 10^{-7}$	0.46
	NW	57.63	72.91	1.27	9.60	921.59	96.00	$4.7 \cdot 10^{-7}$	$4.8 \cdot 10^{-7}$	0.55
	NE	74.85	94.99	1.27	32.77	1 163.07	35.49	$4.9 \cdot 10^{-7}$	$4.6 \cdot 10^{-7}$	0.71
	CAL	93.64	117.93	1.26	60.73	1 442.58	23.75	$5.6 \cdot 10^{-7}$	$4.6 \cdot 10^{-7}$	0.88
	ITA	96.81	120.30	1.24	16.08	1 585.17	98.58	$4.9 \cdot 10^{-7}$	$4.6 \cdot 10^{-7}$	0.94
	DEU	212.27	264.37	1.25	69.28	3 088.06	44.75	$5.4 \cdot 10^{-7}$	$4.8 \cdot 10^{-7}$	1.98
	USA	1 658.4	1 967.24	1.18	529.48	18 270.38	34.50	$6.8 \cdot 10^{-7}$	$5.3 \cdot 10^{-7}$	13.88
400	LUX	1.96	2.19	1.12	1.47	46.76	31.81	$2.6 \cdot 10^{-7}$	$4.5 \cdot 10^{-7}$	0.01
	NY	24.54	31.71	1.29	14.97	403.36	26.94	$6.5 \cdot 10^{-7}$	$5.7 \cdot 10^{-7}$	0.11
	BAY	28.33	36.30	1.28	13.90	490.22	35.27	$5.4 \cdot 10^{-7}$	$5.7 \cdot 10^{-7}$	0.13
	COL	38.41	50.56	1.32	7.27	664.77	91.44	$4.5 \cdot 10^{-7}$	$5.8 \cdot 10^{-7}$	0.17
	DNK	44.66	56.75	1.27	18.47	715.81	38.76	$4.6 \cdot 10^{-7}$	$5.8 \cdot 10^{-7}$	0.21
	FLA	99.92	127.31	1.27	42.71	1 633.26	38.24	$6.0 \cdot 10^{-7}$	$5.9 \cdot 10^{-7}$	0.46
	NW	115.23	147.90	1.28	36.33	1 843.18	50.73	$5.3 \cdot 10^{-7}$	$6.0 \cdot 10^{-7}$	0.53
	NE	150.59	194.48	1.29	63.70	2 326.13	36.52	$6.1 \cdot 10^{-7}$	$6.7 \cdot 10^{-7}$	0.73
	CAL	192.78	248.61	1.29	83.07	2 885.15	34.73	$6.3 \cdot 10^{-7}$	$6.4 \cdot 10^{-7}$	0.91
	ITA	192.47	242.78	1.26	49.50	3 170.33	64.05	$5.2 \cdot 10^{-7}$	$6.1 \cdot 10^{-7}$	0.92
	DEU	423.58	539.66	1.27	62.38	6 176.11	99.01	$5.7 \cdot 10^{-7}$	$6.6 \cdot 10^{-7}$	2.02
	USA	3 178.94	3 962.77	1.24	692.82	36 540.75	52.74	$7.5 \cdot 10^{-7}$	$6.9 \cdot 10^{-7}$	13.95

Table 4), the space occupancy of the WHL index, with $b = 400$, is less than 700 MB, which can be considered a negligible amount of memory for modern commodity hardware. On the contrary, the space occupancy of the B2ALL matrix is more than 36 GB of data, roughly 50 times more than the WHL index. Moreover, as shown in Figures 2 (right) and 3 (right) the gap between the two occupancies tends to increase with b, which suggests that the WHL schemes scales better in terms of space occupancy with respect to b. Even more remarkably, the space to store WHL index is often lower than the space occupancy of the input graph (cf. column $|G|$ of Table 1 and column IS_{WHL} of Tables 2–4) and always lower for large graphs and

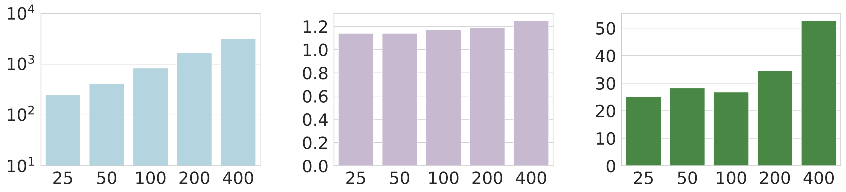


Figure 3 Running time of BuildWhl (left, in seconds); ratio of the running time of BuildWhl to that of BuildBM (middle); ratio of the space occupied by the B2ALL matrix index to that occupied by the WHL (right) for graph USA, as a function of b (x-axis).

values of b. Note that, differently from previously indexing strategies based on the highway labeling [18–20], our scheme is oblivious of the graph topology, that is, once the WHL index is built, algorithm QueryWhl does not need to access the underlying graph to answer BD queries. This is an interesting feature with respect to both space efficiency (the WHL index can be seen as a compressed representation of all pairs beer distances) and usage of the index in a distributed environment (distances can be computed by only accessing the labels of the queried vertices and the distance decoding function [8]).

For the sake of completeness, in Figures 4 and 5, we provide measures of number of label entries stored by algorithm BuildWhl in the labeling part of the computed WHL index. Despite, in the worst case, a WHL stores an entry per beer vertex in each label, here we observe that, in practice, the amount of entries is generally around two orders of magnitude lower than such a worst case.

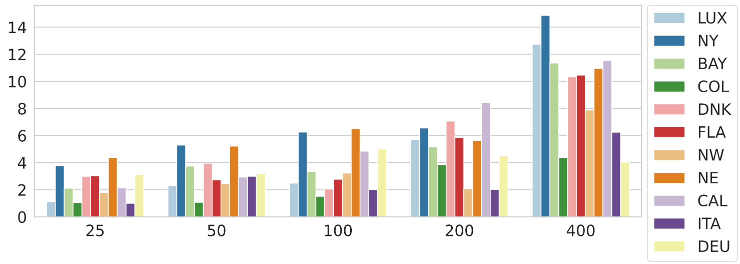


Figure 4 Average number of label entries per vertex (y-axis) stored by algorithm BuildWhl into the labeling part of the WHL index, for each of the considered graphs, except USA, as a function of the number b of beer vertices (x-axis).

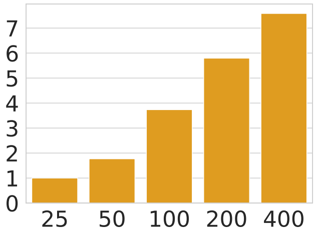


Figure 5 Average number of label entries per vertex (y-axis) stored by algorithm BuildWhl into the labeling part of the WHL index, for graph USA, as a function of the number b of beer vertices (x-axis)

Such behavior is reflected into the low space occupancy requirements discussed above and shows how our preprocessing algorithm, and the highway labeling properties, allow to compute compact representations of shortest beer paths. In this regard, we leave the problem of evaluating whether such effectiveness is influenced by the centrality of beer vertices open for future investigation.

Concerning query times, our experiments highlight that both QueryWhl and QueryBM are extremely fast at answering BD queries (few hundreds of nanoseconds even for large graphs and values of b, cf. column "Query" of Tables 2-4). As expected, QueryWhl is slightly slower than QueryBM, since the query algorithm must consider distances between beer vertices (cf. second term of Eq. 1). Nonetheless, both strategies are competitive under such measure and suited for the requirements of modern data-intensive applications. Our data also confirm the poor performance of BaseLine in this sense: the algorithm does not scale well with both the graph size and b. Indeed, BaseLine's average running time ranges from around 0.01 seconds, on the smallest considered graph and value of b, to 14 seconds on the largest graph and value of b (see Tables 2-4, cf. column BaseLine). In order to give further insights on the effectiveness of employing a whl index for BD queries, in what follows, we provide a cumulative analysis that compares BaseLine, the best known approach without indexing, and our method. In particular, we design and run an experiment where, on the one hand, we execute BaseLine to solve 10^5 queries for randomly selected vertex pairs and measure the total running time. We call this quantity CMT_{BSL} the cumulative running time of BaseLine. On the other hand, we execute BuildWhl to build a whl index and run QueryWhl to answer the same set of queries. We measure the preprocessing time and sum it to the time for executing QueryWhl for all queries. We call this quantity CMT_{WHL} the cumulative running time of whl. The purpose of this experiment is to assess whether the time taken by BuildWhl to construct the index is amortized by the time saved by running QueryWhl to answer BD queries instead of BaseLine, hence to determine the most effective solution in terms of amortized running time.

In Table 5 we present the results of the cumulative experiment for a subset of the considered inputs and values of b. Results for other graphs and values of b are similar and hence omitted. Our data show that the cumulative running time of whl is almost three orders of magnitude lower than that of BaseLine on any combination of graph and number of beer vertices. This is a remarkable result, especially if one considers that the time for precomputing the highway labeling via BuildWhl can be as high as around two hours on usa, and represents a strong experimental evidence of the fact that whl is the most effective framework in practical contexts to answer BD queries, even when large graphs and volumes of queries have to be managed.

Table 5 Cumulative running time of whl and BaseLine to answer to 10^5 BD queries with $b = 200$.

Graph	CMT_{WHL} (s)	CMT_{BSL} (s)	$\text{CMT}_{\text{BSL}}/\ \text{CMT}_{\text{WHL}}$
bay	$1.4 \cdot 10^1$	$1.3 \cdot 10^4$	$9.2 \cdot 10^2$
dnk	$2.2 \cdot 10^1$	$2.0 \cdot 10^4$	$9.3 \cdot 10^2$
cal	$9.3 \cdot 10^1$	$8.8 \cdot 10^4$	$9.4 \cdot 10^2$
ita	$9.6 \cdot 10^1$	$9.4 \cdot 10^4$	$9.7 \cdot 10^2$
deu	$2.1 \cdot 10^2$	$1.9 \cdot 10^5$	$9.3 \cdot 10^2$

Comparison against Frameworks for Generalized Shortest Paths. In this section we complete the experimental evaluation of our framework for BD queries by comparing it with the indexing scheme k-sky of [24], which is considered the best performing method to answer queries on generalized shortest paths, where the aim is computing minimum-weighted paths that traverse at least one vertex for each of k vertex *categories*. Answering to queries on beer distances or shortest beer paths is equivalent to the special case of generalized shortest paths where $k = 1$. For the purpose, method K-SKY preprocesses the graph to store both a 2-hop cover distance labeling and, for each vertex, a so-called *keyword-skyline*. Upon query, the former is exploited to compute quickly shortest paths while the latter, which is a collection of sets of vertices, one per category, is used to reduce the computation of distances toward vertices of each sought category (see [24] for more details). In order to compare K-SKY and WHL, we implemented K-SKY by considering a single POI category (set B) and executed it in the same settings considered for WHL in the previous section. For K-SKY we measure: (i) the running time to build the 2-hop cover distance labeling and to populate the keyword skyline ($\text{PT}_{\text{K-SKY}}$); (ii) the space occupancy to store such two data structures ($\text{IS}_{\text{K-SKY}}$); (iii) the average execution time to answer 10^5 BD queries for randomly generated vertex pairs ($\text{QT}_{\text{K-SKY}}$). For WHL, we measured the same performance indicators discussed in the previous part of the experimentation. Clearly, QueryWhl is run with the same set of queries.

An excerpt of the results of the above experiment is given in Table 6. Data for other graphs and values of b are omitted due to space limitations. The main conclusion that can be drawn from our experimental data is that our method WHL outperforms method K-SKY with respect to all performance metrics. Specifically, the preprocessing time of K-SKY is orders of magnitude larger than the running time of BuildWhl, as it runs for more than one hour even for small inputs. This is expected, since the index constructed by BuildWhl can be seen as a compact version of the 2-hop cover labeling. Also from a space occupancy viewpoint, K-SKY stores at least two orders of magnitude more information with respect to the WHL index. Finally, the average query time offered by K-SKY is always around 3 orders of magnitude larger than that of QueryWhl.

Table 6 Performance of WHL and K-SKY on graphs NY, BAY, COL with $b \in \{50, 100\}$ beer vertices.

b	Graph	**Preprocessing** (s)		**Space Occupancy** (MB)		**Query** (s)	
		PT_{WHL}	$\text{PT}_{\text{K-SKY}}$	IS_{WHL}	$\text{IS}_{\text{K-SKY}}$	QT_{WHL}	$\text{QT}_{\text{K-SKY}}$
50	NY	3.11	> 3600	5.34	795.02	$3.5 \cdot 10^{-7}$	$1.3 \cdot 10^{-4}$
	BAY	3.68	> 3600	4.59	1385.53	$3.3 \cdot 10^{-7}$	$1.6 \cdot 10^{-4}$
	COL	4.77	> 3600	1.78	1982.95	$4.9 \cdot 10^{-7}$	$1.5 \cdot 10^{-4}$
100	NY	6.25	> 3600	6.31	1467.21	$3.7 \cdot 10^{-7}$	$2.6 \cdot 10^{-4}$
	BAY	7.16	> 3600	4.09	2726.54	$3.4 \cdot 10^{-7}$	$3.3 \cdot 10^{-4}$
	COL	9.60	> 3600	2.52	3794.05	$4.1 \cdot 10^{-7}$	$3.0 \cdot 10^{-4}$

5 Conclusion and Future Work

We have showcased that an indexing scheme, to efficiently execute queries on beer distances or shortest beer paths for pairs of vertices of a beer graph, can be built by adapting the highway labeling. Through extensive experimentation on real-world graphs, we have empirically demonstrated its practical effectiveness and superiority, in terms of offered trade-off between preprocessing time, space occupancy and query time, with respect to the state-of-the-art. Our work leaves several questions open for future investigation. First and foremost, it would be interesting to understand whether an indexing method with a space/computational time trade-off better than those mentioned in this paper (either empirically or in the worst case), can be designed for general graphs. Another relevant direction to explore would be extending the experimental comparison of WHL, B2ALL K-SKY given here to digraphs and to queries on shortest beer paths. A more challenging, but certainly of interest, objective to pursue would be designing a dynamic algorithm to maintain the WHL index under changes of the set B.

References

1 Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. 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, USA*, pages 349–360. ACM, 2013. doi:10.1145/2463676.2465315.

2 Shikha Anirban, Junhu Wang, and Md. Saiful Islam. Experimental evaluation of indexing techniques for shortest distance queries on road networks. In *39th IEEE International Conference on Data Engineering (ICDE 2023) Anaheim, USA*, pages 624–636. IEEE, 2023. doi:10.1109/ICDE55515.2023.00054.

3 Joyce Bacic, Saeed Mehrabi, and Michiel Smid. Shortest beer path queries in outerplanar graphs. In Hee-Kap Ahn and Kunihiko Sadakane, editors, *32nd International Symposium on Algorithms and Computation (ISAAC 2021), Fukuoka, Japan*, volume 212 of *LIPIcs*, pages 62:1–62:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ISAAC.2021.62.

4 Joyce Bacic, Saeed Mehrabi, and Michiel Smid. Shortest beer path queries in outerplanar graphs. *Algorithmica*, 85(6):1679–1705, 2023. doi:10.1007/S00453-022-01045-4.

5 Valentin Buchhold, Dorothea Wagner, Tim Zeitz, and Michael Zündorf. Customizable Contraction Hierarchies with Turn Costs. In Dennis Huisman and Christos D. Zaroliagis, editors, *20th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2020)*, volume 85 of *Open Access Series in Informatics (OASIcs)*, pages 9:1–9:15, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/OASIcs.ATMOS.2020.9.

6 Alessio Cionini, Gianlorenzo D'Angelo, Mattia D'Emidio, Daniele Frigioni, Kalliopi Giannakopoulou, Andreas Paraskevopoulos, and Christos D. Zaroliagis. Engineering graph-based models for dynamic timetable information systems. In Stefan Funke and Matús Mihalák, editors, *14th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems, ATMOS 2014, September 11, 2014, Wroclaw, Poland*, volume 42 of *OASIcs*, pages 46–61. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2014. doi:10.4230/OASICS.ATMOS.2014.46.

7 Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. Reachability and distance queries via 2-hop labels. *SIAM J. Comput.*, 32(5):1338–1355, 2003. doi:10.1137/S0097539702403098.

8 Feliciano Colella, Mattia D'Emidio, and Guido Proietti. Simple and practically efficient fault-tolerant 2-hop cover labelings. In Dario Della Monica, Aniello Murano, Sasha Rubin, and Luigi Sauro, editors, *Joint Proceedings of the 18th Italian Conference on Theoretical Computer Science (ICTCS 2017) and the 32nd Italian Conference on Computational Logic (CILC 2017), Naples, Italy*, volume 1949 of *CEUR Workshop Proceedings*, pages 51–62. CEUR-WS.org, 2017.

9 David Coudert, Andrea D'Ascenzo, and Mattia D'Emidio. D-hash/ShortestBeerDistanceQueries. Software (visited on 2024-09-20). URL: https://github.com/D-hash/ShortestBeerDistanceQueries.

10 Gianlorenzo D'Angelo, Mattia D'Emidio, and Daniele Frigioni. Fully dynamic 2-hop cover labeling. *ACM J. Exp. Algorithmics*, 24(1):1.6:1–1.6:36, 2019. doi:10.1145/3299901.

11 Rathish Das, Meng He, Eitan Kondratovsky, J. Ian Munro, Anurag Murty Naredla, and Kaiyu Wu. Shortest beer path queries in interval graphs. In Sang Won Bae and Heejin Park, editors, *33rd International Symposium on Algorithms and Computation (ISAAC 2022), Seoul, Korea*, volume 248 of *LIPIcs*, pages 59:1–59:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ISAAC.2022.59.

12 Andrea D'Ascenzo and Mattia D'Emidio. Top-k distance queries on large time-evolving graphs. *IEEE Access*, 11:102228–102242, 2023. doi:10.1109/ACCESS.2023.3316602.

13 Daniel Delling, Julian Dibbelt, Thomas Pajor, and Tobias Zündorf. Faster Transit Routing by Hyper Partitioning. In Gianlorenzo D'Angelo and Twan Dollevoet, editors, *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS*

2017), volume 59 of *Open Access Series in Informatics (OASIcs)*, pages 8:1–8:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/OASIcs.ATMOS.2017.8.

14 Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck. Robust distance queries on massive networks. In Andreas S. Schulz and Dorothea Wagner, editors, *Proceedings of the 22th Annual European Symposium on Algorithms (ESA 2014), Wroclaw, Poland*, volume 8737 of *Lecture Notes in Computer Science*, pages 321–333. Springer, 2014. doi:10.1007/978-3-662-44777-2_27.

15 Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Connection scan algorithm. *ACM J. Exp. Algorithmics*, 23, 2018. doi:10.1145/3274661.

16 Edsger W. Dijkstra. A note on two problems in connexion with graphs. In Krzysztof R. Apt and Tony Hoare, editors, *Edsger Wybe Dijkstra: His Life, Work, and Legacy*, volume 45 of *ACM Books*, pages 287–290. ACM / Morgan & Claypool, 2022. doi:10.1145/3544585.3544600.

17 Jochen Eisner and Stefan Funke. Sequenced route queries: getting things done on the way back home. In *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*, SIGSPATIAL 2012, pages 502–505, New York, NY, USA, 2012. ACM. doi:10.1145/2424321.2424400.

18 Muhammad Farhan, Koehler Henning, and Qing Wang. BatchHL$^+$: batch dynamic labelling for distance queries on large-scale networks. *The VLDB Journal*, pages 1–29, 2023. doi:10.1007/s00778-023-00799-9.

19 Muhammad Farhan and Qing Wang. Efficient maintenance of highway cover labelling for distance queries on large dynamic graphs. *World Wide Web (WWW)*, 26(5):2427–2452, 2023. doi:10.1007/S11280-023-01146-2.

20 Muhammad Farhan, Qing Wang, Yu Lin, and Brendan D. McKay. A highly scalable labelling approach for exact distance queries in complex networks. In Melanie Herschel, Helena Galhardas, Berthold Reinwald, Irini Fundulaki, Carsten Binnig, and Zoi Kaoudi, editors, *Proceedings of 22nd International Conference on Extending Database Technology (EDBT 2019), Lisbon, Portugal*, pages 13–24. OpenProceedings.org, 2019. doi:10.5441/002/EDBT.2019.03.

21 Roy Friedman and Alex Kogan. Deterministic dominating set construction in networks with bounded degree. In Marcos K. Aguilera, Haifeng Yu, Nitin H. Vaidya, Vikram Srinivasan, and Romit Roy Choudhury, editors, *Distributed Computing and Networking*, pages 65–76, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

22 Joachim Gudmundsson and Yuan Sha. Shortest beer path queries in digraphs with bounded treewidth. In Satoru Iwata and Naonori Kakimura, editors, *34th International Symposium on Algorithms and Computation, ISAAC 2023, December 3-6, 2023, Kyoto, Japan*, volume 283 of *LIPIcs*, pages 35:1–35:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ISAAC.2023.35.

23 Tesshu Hanaka, Hirotaka Ono, Kunihiko Sadakane, and Kosuke Sugiyama. Shortest beer path queries based on graph decomposition. In Satoru Iwata and Naonori Kakimura, editors, *34th International Symposium on Algorithms and Computation (ISAAC 2023), Kyoto, Japan*, volume 283 of *LIPIcs*, pages 37:1–37:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ISAAC.2023.37.

24 Vassilis Kaffes, Alexandros Belesiotis, Dimitrios Skoutas, and Spiros Skiadopoulos. Finding shortest keyword covering routes in road networks. In *Proceedings of the 30th International Conference on Scientific and Statistical Database Management*, pages 1–12, 2018.

25 Vassilissa Lehoux and Christelle Loiodice. Faster preprocessing for the trip-based public transit routing algorithm. In Dennis Huisman and Christos D. Zaroliagis, editors, *20th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems, ATMOS 2020, September 7-8, 2020, Pisa, Italy (Virtual Conference)*, volume 85 of *OASIcs*, pages 3:1–3:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/OASICS.ATMOS.2020.3.

26 Jure Leskovec and Rok Sosič. Snap: A general-purpose network analysis and graph-mining library. *ACM Trans. Intell. Syst. Technol.*, 8(1), July 2016. doi:10.1145/2898361.

27 Huiping Liu, Cheqing Jin, Bin Yang, and Aoying Zhou. Finding top-k optimal sequenced routes. In *34th IEEE International Conference on Data Engineering, ICDE 2018, Paris, France, April 16-19, 2018*, pages 569–580. IEEE Computer Society, 2018. doi:10.1109/ICDE.2018.00058.

28 Catherine C. McGeoch. *A Guide to Experimental Algorithmics.* Cambridge University Press, 2012.

29 Michael N. Rice and Vassilis J. Tsotras. Engineering generalized shortest path queries. In Christian S. Jensen, Christopher M. Jermaine, and Xiaofang Zhou, editors, *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*, pages 949–960. IEEE Computer Society, 2013. doi:10.1109/ICDE.2013.6544888.

30 Mehdi Sharifzadeh, Mohammad Kolahdouzan, and Cyrus Shahabi. The optimal sequenced route query. *The VLDB journal*, 17:765–787, 2008.

31 Chao Zhang, Angela Bonifati, and M. Tamer Özsu. An overview of reachability indexes on graphs. In *Companion of the 2023 International Conference on Management of Data*, SIGMOD 2023, pages 61–68, New York, NY, USA, 2023. ACM. doi:10.1145/3555041.3589408.

32 Junhua Zhang, Long Yuan, Wentao Li, Lu Qin, Ying Zhang, and Wenjie Zhang. Label-constrained shortest path query processing on road networks. *The VLDB Journal*, 33:569–593, 2024. doi:10.1007/s00778-023-00825-w.

33 Yikai Zhang and Jeffrey Xu Yu. Hub labeling for shortest path counting. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD 2020, pages 1813–1828, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3318464.3389737.

34 Zibin Zheng, Fanghua Ye, Rong-Hua Li, Guohui Ling, and Tan Jin. Finding weighted k-truss communities in large networks. *Information Sciences*, 417:344–360, 2017. doi:10.1016/j.ins.2017.07.012.

35 Huaijie Zhu, Wenbin Li, Wei Liu, Jian Yin, and Jianliang Xu. Top k optimal sequenced route query with POI preferences. *Data Sci. Eng.*, 7(1):3–15, 2022. doi:10.1007/S41019-022-00177-5.

Pricing for the EVRPTW with Piecewise Linear Charging by a Bounding-Based Labeling Algorithm

Jenny Enerbäck
Department of Mathematics, Linköping University, Sweden
Scania CV AB, Södertälje, Sweden

Lukas Eveborn[1] ✉ 🏠
Department of Mathematics, Linköping University, Sweden

Elina Rönnberg ✉ 🏠
Department of Mathematics, Linköping University, Sweden

Abstract

The elementary shortest path problem with resource constraints (ESPPRC) is a common problem that often arises as a pricing problem when solving vehicle routing problems with a column generation approach. One way of solving the ESPPRC is to use a labeling algorithm. In this paper, we focus on how different bounding strategies for labeling algorithms can be adapted and strengthened for the ESPPRC that arises from the Electric Vehicle Routing Problem with Time Windows and Piecewise Linear Recharging function (EVRPTW-PLR). We present a new completion bound method that takes charging times into account, and show how the completion bound can be combined with ng-routes. Computational experiments show that the new completion bound combined with ng-routes significantly improves the performance compared to a basic labeling algorithm.

2012 ACM Subject Classification Mathematics of computing → Combinatorics; Mathematics of computing → Graph algorithms; Applied computing → Transportation; Mathematics of computing → Mathematical software

Keywords and phrases ESPPRC, EVRP, Bounding, Labeling Algorithm

Digital Object Identifier 10.4230/OASIcs.ATMOS.2024.3

Supplementary Material
Software (Public Repo): `https://gitlab.liu.se/eliro15/labeling-algorithm-for-evrptw-plr`

Funding This work was funded by the Swedish Energy Agency within the program FFI, Fordonsstrategisk Forskning och Innovation, under the grant Condore (P2022-00952).
Lukas Eveborn: This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

Acknowledgements The Condore project is an important platform for this research as it provides close collaborations with the industrial partners Scania and Ragn-Sells as well as the Vehicular Systems division at Linköping University. We also want to thank the reviewers for useful feedback and especially the reviewer that pointed out an issue related to the combination of ng-routes and bounding.

1 Introduction

The interest in using electrical vehicles for transportation has been steadily increasing in the last years. One of the main reasons is the positive environmental aspects compared to using traditional combustion vehicles. As for the traditional vehicles, the routing and scheduling of the electric vehicles is a crucial aspect to consider. This has created a new type of vehicle routing problem, the Electric Vehicle Routing Problem (EVRP).

[1] Corresponding author

24th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2024).
Editors: Paul C. Bouman and Spyros C. Kontogiannis; Article No. 3; pp. 3:1–3:18
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

As for traditional routing problems, a common way of solving the EVRP is to use a column generation approach. Given a path representation of the routes, the column generation pricing problem can be represented as a shortest path problem. In such a setting, the performance of the pricing problem is crucial for the overall performance of the column generation approach. In this paper we are interested in the pricing problem for the EVRP with Time Windows, Piecewise Linear Recharging function and partial recharging (EVRPTW-PLR). This pricing problem can be represented as an elementary shortest path problem with resource constraints (ESPPRC). The ESPPRC can be solved in several ways, one of them is to use a labeling algorithm [9] which is the method we will be using.

Compared to traditional routing problems, the electrical routing problems come with a new set of challenges, the main one being the need to plan for recharging of the vehicles. This changes the characteristics of the ESPPRC and the labeling algorithm needs to be adapted to handle this, something that has been previously studied (see e.g. [5]). Another challenge is to model the recharging realistically, for which a piecewise linear function has been proven to be a good choice [11]. This does further complicate the ESPPRC and necessitates adaptions of the labeling algorithm, as done in e.g. [10] and [2].

How well a labeling algorithm performs is highly dependent on the different acceleration strategies that are used. Some of the most common acceleration strategies are bidirectional labeling [14], ng-routes [1] and bounding (see e.g. [3, 14]). The implementation of ng-routes is not affected by the piecewise linear charging function and can easily be implemented in a standard way. For bidirectional labeling, the case is different, as the standard strategy cannot be applied for piecewise linear charging functions. To the authors' best knowledge, this has not yet been done, and we find it hard to see how it can be done efficiently. For this reason, we consider bounding to be an important type of acceleration strategy in labeling algorithms for piecewise linear charging functions. Bounding builds on the idea to discard unpromising labels. There are several ways to do this, one is the resource bounding, which is a way to discard labels that cannot be completed to a feasible solution. Another way is the completion bounding, which is a way to discard labels that cannot be completed to a better solution than the best found so far.

In this paper, we focus on how different bounding strategies can be adapted and strengthened for the ESPPRC that arises from the EVRPTW-PLR. The paper extends the result of the master thesis [7] that introduced a bounding-based labeling algorithm for the EVRPTW with linear and partial recharging. The main contributions of this paper are:

- A new time-based completion bound method that takes charging times into account.
- Integration of completion bounds and ng-routes that handles the assumption of elementarity in completion bounds with the relaxation of elementarity in ng-paths.
- Publicly available labeling algorithm solving the ESPPRC for the EVRPTW-PLR (available at: `https://gitlab.liu.se/eliro15/labeling-algorithm-for-evrptw-plr`).

The rest of this paper is structured as follows. In Section 2, the problem statement and model will be presented. A short explanation of the foundations of the labeling algorithm will be presented in Section 3. The implemented bounding methods will then be presented in Section 4. Experimental results will be presented in Section 5 comparing the effect of the different bounding methods implemented, and finally the conclusions are presented in Section 6.

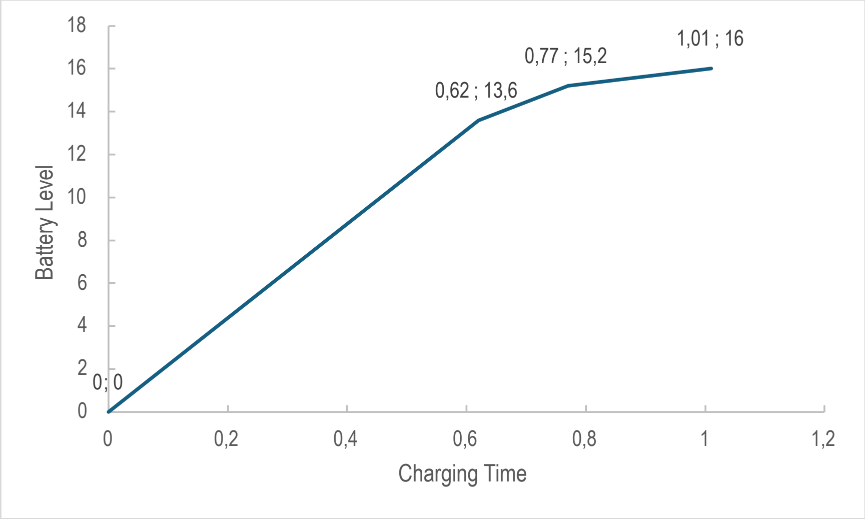


Figure 1 The piecewise linear charging curve used in this paper, with data from [11].

2 Problem Statement and Model

The problem considered is the ESPPRC that arises as a pricing problem from solving the EVRPTW-PLR within a column generation approach. In the EVRPTW-PLR, a fleet of electric vehicles with limited load and battery capacity should visit a set of customers, where each customer has a given time window and a given demand, while minimizing the total cost. Furthermore, the vehicles start and end at a depot, and can recharge at charging stations. The recharging rate is given by a piecewise linear function.

Inspired by the notation used in [5], we formulate the ESPPRC on a directed graph $G(V, A)$, where V denotes the set of nodes, and A denotes the set of arcs connecting the nodes. The set of nodes V consists of customer nodes, given by the set N, charging nodes, given by the set R, a start node o, and an end node d.

An arc between node $i \in V$ and node $j \in V$ is associated with three parameters: the cost c_{ij} of the arc, the energy consumption r_{ij} for traversing the arc, and the travel time t_{ij}, where also the service time at i is included if $i \in N$, i.e. i is a customer node. We assume that the triangle inequality holds for the time and energy consumption, i.e. $t_{ij} \leq t_{ik} + t_{kj}$ and $r_{ij} \leq r_{ik} + r_{kj}$ for all $i, j, k \in V$. Each customer node $i \in N$ is associated with a service time window $[e_i,\ l_i]$, during which the service can start. The vehicle is allowed to arrive early at a customer but has to wait until the start time to start service. Each customer node $i \in N$ is associated with a demand q_i, which since we are solving the pricing problem, only needs to be satisfied if the customer is visited.

The vehicle has a maximum load C and a maximum battery capacity Q. It can recharge at charging nodes where partial recharging is allowed. The battery level cannot go below 0.

The charging curve is given by a piecewise linear function, and we define it as in [10] where the charging curve is given by a set of pieces $P = \{1 \ldots W\}$. Each piece $p \in P$ is defined by a start and end battery level τ_{p-1} and τ_p, and a recharging rate ρ_p (energy per timestep). In reality, the charging function is often concave since the charging rate decreases as the battery level increases [11]. With these assumptions, it holds that $0 = \tau_0 \leq \tau_1 \leq \ldots \leq \tau_W = Q$ and $\rho_1 \geq \rho_2 \geq \ldots \geq \rho_W > 0$. An example is shown in Figure 1.

The vehicle starts at the start node at time 0 with a full battery and must return to the end node before the max time, $T_{\max}$. It can visit a customer node at most once, while charging nodes can be visited multiple times. The objective is to find a route that minimizes the total cost of the arcs travelled, while satisfying the constraints. Because we are in a column generation setting, the arc costs are in fact reduced costs and as such they can become negative. In the EVRPTW-PLR, we assume the initial costs to be positive, and customers to be associated with dual variables, leading to that the outgoing arcs from customers can become negative, while the other arcs have their initial cost.

3 Labeling Algorithm

The basic version of the labeling algorithm is presented in Algorithm 1. The algorithm is implemented with a priority queue Γ of labels waiting to be extended and continues until there is no label to be extended from this queue. In order to find a good complete path early, the priority queue is ordered by increasing costs and the label with the lowest cost is first extended. This is especially important when using completion bounds, since the strength of the completion bounds depend on the quality of the incumbent upper bound. A label is denoted by L_i and represents a partial path $o \rightarrow i \in N$. Each label contains a number of resources, such as the cost of the partial path and the battery level. When extending a label L_i to a node j the function $\mathit{Extend}(L_i, j)$ updates the label resources given by the defined Resource Extension Functions (REFs) and verifies the feasibility of the extension of the path. All neighbours that can be reached from node i are denoted by Δ_i. The function $\mathit{Dominance}(\Lambda_j, L_j)$ checks dominance between the new label L_j and all labels in Λ_j, which is a bucket containing all labels ending at node j, given the decided dominance criteria. All labels that L_j dominates are discarded, and the $\mathit{Dominance}(\Lambda_j, L_j)$ returns false if L_j is dominated, whereupon L_j is discarded. The shortest path can, at the end of execution, be selected from the set of labels at the end node.

Algorithm 1 Basic labeling algorithm.

```
// Initialization of priority queue $\Gamma$ with a start label $\Lambda_o$
$\Gamma \leftarrow \Lambda_o$
while $\Gamma \neq \emptyset$ do
    // Get next label to extend
    $L_i \leftarrow \text{pop}(\Gamma)$
    // Try to extend label to all outgoing neighbours of $i$
    for $j \in \Delta_i$ do
        // Try to extend label to node $j$
        if $L_j \leftarrow \mathit{Extend}(L_i, j)$ then
            // If label finishes at end node, it is not added to priority queue
            if $j = d$ then
                $\Lambda_d \leftarrow L_j$
            // Else check dominance between new label and all labels at node $j$
            else if $\mathit{Dominance}(\Lambda_j, L_j)$ then
                $\Gamma \leftarrow L_j$
                $\Lambda_j \leftarrow L_j$
            end
        end
    end
end
```

The resources of label L_i are defined as suggested by Lam et al. [10], with the notation of Desaulnier et al. [5], and presented below.

T_i^{cost}: Cost of the partial path $o \to i$.

T_i^{load}: Delivered load along the partial path.

T_i^{time}: Earliest service start time at node i that ensures time window and battery feasibility along the path.

T_i^{energy}: Battery level at i, assuming that minimal recharges have been performed at all visited charging stations.

$T_i^{\text{mrt}_p}$: Available charging time at each charging piece $p \in P$ that can be added at previous charging stops while ensuring time window feasibility.

$T_i^{\text{cust}_n}$: Indicates if a customer $n \in N$ cannot be visited in the extension of the path. This can either be because they already are visited or because they are unreachable from the path ending at node j.

A label L_i at node $i \in N$ is extended to a node $j \in N$ along the arc $(i, j) \in A$ by a number of Resource Extension Functions (REFs) that update the resources of the label, creating a new label L_j. The REFs are defined as suggested by Lam et al. [10], with the notation of Desaulnier et al. [5]. The REFs for the cost of the partial path and the delivered load are defined by (1) and (2).

$$T_j^{\text{cost}} = T_i^{\text{cost}} + c_{ij} \tag{1}$$

$$T_j^{\text{load}} = T_i^{\text{load}} + q_j \tag{2}$$

To update the time of the partial path and the energy level, we need to define some new concepts. Whenever a vehicle can arrive early at a customer, i.e. $T_i^{\text{time}} + t_{ij} < e_j$, a slack time is introduced, denoted W_{ij}. This slack time is given by $W_{ij} = \max\{0, e_j - T_i^{\text{time}} - t_{ij}\}$, which represents how much later it is possible to leave node i without delaying service start time at node j. If charging stations have been visited previous to node j, the slack time can be used for charging to the extent allowed by the available charging time $T_i^{\text{mrt}_p}$, $p \in P$. We assume that all available slack time, in the extent that it is possible, is used for charging and denote the energy charged during slack time by S_{ij}. Since the charging curve is piecewise linear, we have several charging pieces $p \in P$. In order to calculate S_{ij}, we therefore need to calculate the amount of time we can recharge at each charging piece, which we denote by $\delta_{pij}^{\text{slack}}$. This is calculated for each piece $p \in P$ in decreasing order of charging rate by $\delta_{pij}^{\text{slack}} = \max\{0, \min\{W_{ij} - \sum_{\mu=1}^{p-1} \delta_{\mu ij}^{\text{slack}}, T_i^{mrt_p}\}\}$. In this formula $T_i^{mrt_p}$ is, as mentioned, the available charging time at charging piece p from previous charging stops, and $W_{ij} - \sum_{\mu=1}^{p-1} \delta_{\mu ij}^{\text{slack}}$ is the remaining available slack time for charging at piece p, given the charging time at previous, more advantageous, pieces. The energy charged during slack time is then given by $S_{ij} = \sum_{p \in P} \rho_p \delta_{pij}^{\text{slack}}$ where ρ_p is the charging rate at piece p.

If the energy charged during slack time together with the initial energy is not enough to cover the energy consumption r_{ij} of the arc (i, j), i.e. $T_i^{\text{energy}} - r_{ij} + S_{ij} < 0$, additional charging is required. The required additional charging energy is denoted by Z_{ij} and the time for this additional charging is denoted X_{ij}. Similarly to how the energy charged during slack time was calculated, we need to calculate the amount of time $\delta_{pij}^{\text{extra}}$ we recharge at each charging piece $p \in P$. This is calculated for each piece $p \in P$ in decreasing order of charging rate by $\delta_{pij}^{\text{extra}} = \max\{0, \min\{T_i^{mrt_p} - \delta_{pij}^{\text{slack}}, (-T_i^{\text{energy}} + r_{ij} - S_{ij} - \sum_{\mu=1}^{p-1} \rho_\mu \delta_{\mu ij}^{\text{extra}})/\rho_p\}\}$. In this formula $T_i^{mrt_p} - \delta_{pij}^{\text{slack}}$ is the remaining available charging time for charging at piece p after slack charging has been done and $(-T_i^{\text{energy}} + r_{ij} - S_{ij} - \sum_{\mu=1}^{p-1} \rho_\mu \delta_{\mu ij}^{\text{extra}})/\rho_p$ is the remaining time, given the piece p charging rate, needed to be recharged after slack charging has been done and additional charging has been done at previous, more advantageous, pieces. The additional charging time and energy are then given by $X_{ij} = \sum_{p \in P} \delta_{pij}^{\text{extra}}$ and $Z_{ij} = \sum_{p \in P} \delta_{pij}^{\text{extra}} \rho_p$, respectively.

The REF for time can now be given in (3), where, as stated, X_{ij} is the required additional charging time and e_j is the earliest allowed arrival at node j. The energy level is updated according to the REF defined in (4), where, as stated, r_{ij} is the energy consumption of the arc, S_{ij} is the energy charged during slack time, and Z_{ij} is the additional energy charged.

$$T_j^{\text{time}} = \max\{T_i^{\text{time}} + t_{ij} + X_{ij}, e_j\} \tag{3}$$

$$T_j^{\text{energy}} = T_i^{\text{energy}} - r_{ij} + S_{ij} + Z_{ij} \tag{4}$$

The REF for the available charging time is calculated for each piece $p \in P$ in decreasing order of charging rate, where τ_p is the end battery level of piece $p \in P$. It is given in (5) and calculated differently depending on whether j is a charging station or not. If a charging station, the time is calculated as the minimum of the maximum charging time at the piece p and the time needed to reach the end battery level at the piece p given current battery level T_j^{energy}. If not a charging station, it is calculated as the minimum of available charging time, after slack and additional charging has been added, and available time until the end of the time window after available charging time at more advantageous pieces has been added.

$$T_j^{\text{mrt}_p} = \begin{cases} \min\{(\tau_p - \tau_{p-1})/\rho_p, \max\{(\tau_p - T_j^{\text{energy}})/\rho_p, 0\}\}, & \text{if } j \in R \\ \min\{T_i^{\text{mrt}_p} - \delta_{pij}^{\text{slack}} - \delta_{pij}^{\text{extra}}, l_j - T_j^{\text{time}} - \sum_{\mu=1}^{p-1} T_j^{\text{mrt}_\mu}\}, & \text{if } j \in V \setminus R \end{cases} \tag{5}$$

Finally, customer reachability is updated in REF (6), either if they are visited or if they are unreachable from the path ending at node j. A node $n \in N \setminus \{j\}$ is marked as unreachable from the path ending at node j in the function $R(T_j^{\text{load}}, T_j^{\text{time}})$ by the value 1 if $T_j^{\text{load}} + q_n > C$ or $T_j^{\text{time}} + t_{jn} > l_n$ holds. This method of marking unreachable nodes as already visited was initially suggested by Feillet et al. [8].

$$T_j^{\text{cust}_n} = \begin{cases} T_i^{\text{cust}_n} + 1, \text{ if } j = n, \\ \max\{T_i^{\text{cust}_n}, R(T_j^{\text{load}}, T_j^{\text{time}})\} \; j \in N \setminus \{n\} \end{cases} \tag{6}$$

After the resources of the label have been updated, the feasibility of the extension is verified. As defined by [10], an extended label is feasible if the following four inequalities hold: $T_j^{\text{load}} \leq C$, $T_j^{\text{time}} \leq l_j$, $T_j^{\text{energy}} \geq 0$, and $(T_j^{\text{cust}_n}) \leq 1$ for all $n \in N$.

Once a label has been feasibly extended to node j, dominance is checked on the other labels finishing in node j. For a label $\tilde{L}_j$ at j to dominate another label $\hat{L}_j$ at j, the criteria given by Equations (7) through (11) must hold as defined by [10]. If $\tilde{L}_j$ dominates $\hat{L}_j$, $\hat{L}_j$ is discarded.

$$\tilde{T}_j^{\text{cost}} \leq \hat{T}_j^{\text{cost}} \tag{7}$$

$$\tilde{T}_j^{\text{load}} \leq \hat{T}_j^{\text{load}} \tag{8}$$

$$\tilde{T}_j^{\text{time}} \leq \hat{T}_j^{\text{time}} \tag{9}$$

$$\tilde{T}_j^{\text{cust}_n} \leq \hat{T}_j^{\text{cust}_n}, \; n \in N \tag{10}$$

$$\tilde{f}_j^{\text{energy}}(t) \geq \hat{f}_j^{\text{energy}}(t), \; t \in [\hat{T}_j^{\text{time}}, l_j] \tag{11}$$

In Equation (11), battery levels are compared given a later arrival time at j with the help of the function $f_j^{\text{energy}}(t)$ which outputs the battery level given an arrival time t. This has to be done because T_j^{energy} is only the energy level given a *minimal* recharge, but since the extensions are unknown at the time of dominance checking, it might be needed to arrive later but with a higher battery level. So in order for the label $\tilde{L}_j$ to dominate $\hat{L}_j$, it has to be able to achieve a higher battery level for every possible arrival time at node j of $\hat{L}_j$. The possible arrival times are all the times between $\hat{T}_j^{\text{time}}$ and the latest service start time at node j, l_j. However, since the charging curves are piecewise linear, it suffices to verify this at start points, breakpoints and endpoints of the charging curves. For a more thorough explanation and definition of this function, we refer to Lam et al. [10].

4 Bounding Methods

A well-known challenge when applying labeling algorithms is that in the late iterations, a huge number of labels have been generated and a large portion of those cannot be completed into a solution of interest. To avoid this, or at least discard some unpromising ones along the way, bounding methods can be used. Bounding methods use optimistic bounds on the completion of a path to discard labels that cannot yield a best solution. This can be done in different ways, but common for all is that the methods need to rely on fast computations to contribute to computational efficiency. For that reason the methods can often be greedy and do not necessarily use the network structure or all information in the problem. Furthermore, it is often important to use the problem structure to make the bounding methods even more efficient. We choose to implement two types of bounding methods, resource bounds and completion bounds, to improve the computational time of the labeling algorithm. These are presented in Section 4.1 and 4.2, respectively.

4.1 Resource Bounds

The main idea behind resource bounds is that if there is not enough resources left to reach the end node, a label can be discarded – even if there still are resource feasible extensions to some nodes. We implement resource bounds for battery feasibility and time feasibility, as they are the only resources that directly can prevent feasible completions of a path. The resource bounds are implemented in the $\mathit{Extend}(L_i, j)$ function.

When formulating our resource bounds, inspiration is taken from the criterion $W^r + w_{ij}^r + \underline{w}_{jd}^r > W_{\max}^r$ suggested by Boland et al. [3], where W^r is the consumption of resource r along the partial path, w_{ij} the consumption on the arc (i, j), and $\underline{w}_{jd}^r$ a lower bound on the resource consumption from node j to the depot.

Starting with battery feasibility, it is checked that the partial path can reach the end node or a charger without emptying the battery. For a node $j \in V$, let $r_{j\xi}$ be the energy consumption to travel from j to the closest node that is either a charger or the end node, i.e. $\xi \in R \cup \{d\}$. If the current battery level together with the available charging time is not enough to reach ξ, the label can be discarded. This is done by checking if $T_j^{\text{energy}} + \sum_{p \in P} \rho_p T_j^{\text{mrt}_p} - r_{j\xi} < 0$.

For time feasibility, we generate an optimistic bound on the time it takes to reach the end node and use this to check that the end node can be reached within the max time limit. Beyond travel time we also consider possible extra charging time that is needed to reach the end node. In order to make sure it is an optimistic bound, i.e. no labels are discarded wrongly, it is assumed that charging can be done "on the road" i.e. discarding the possible extra travel time and energy it takes to reach a charging station. For the same reason it is also assumed that the recharge is done at the fastest charging rate, ρ_1. To formulate the condition, we first need to calculate the possible required additional energy to reach the end node, which is given by $Y_j = \max\{0, r_{jd} - T_j^{\text{energy}}\}$, where r_{jd} is the energy consumption to reach the depot from current node $j \in N$. The feasibility check can then be done by $T_j^{\text{time}} + t_{jd} + Y_j/\rho_1 > T_{\text{end}}$. If the inequality holds, the label is discarded.

4.2 Completion Bounds

For completion bounds, the idea is to discard labels that are guaranteed to not be completed to a better solution than the current best solution. This is done by generating an optimistic bound on the cost of completion of the path, denoted z_{LB}. A label L_j with current cost T_j^{cost}, can then be discarded if $T_j^{\text{cost}} + z_{\text{LB}} > UB$, where UB is the current upper bound on the cost for a finished path. We choose to calculate two completion bounds, one by solving a knapsack problem with respect to the vehicle load constraint and one by solving a problem that takes both travel time and charging time into account. Both are inspired by the knapsack bound formulation of Righini and Salani [14].

For the load-based bound, the implementation is similar to the formulation of [14], but with some adjustments to our problem. They do not have charging nodes, but as it turns out, the charging nodes can be disregarded in the calculation of the bound. The reason for this is twofold, firstly, the battery constraints are relaxed in the bound, so visits to chargers are not required for feasibility reasons. Secondly, in a standard column generation context, the dual variables of the charging nodes are non-existent as there are no master constraints related to them, and it can therefore never be profitable to visit chargers assuming positive initial costs. With this in mind, let then S be the set of already visited customers. We can then define the continuous decision variable $y_k \in [0, 1],\ k \in V \setminus S$ that state how much a customer is visited.

We define the cost of visiting a node $i \in N$ as the cost of its cheapest outgoing arc, i.e. $u_i = \min_{l \in N \setminus R} c_{il}$. This is used to get the cost of visiting a customer in the case when $i \in N \setminus S$ which we only would want to do if u_i is negative. It is also used to get the cost of leaving the current node j in the case when $i = j$, which is necessary to get the right number of arcs in the solution. Compared to [14], we consider the outgoing arcs from the nodes $k \in N$ instead of the incoming arcs to calculate the least cost since the service time in our case is included in the outgoing arcs and not the incoming arcs. Note also that to calculate the cost for visiting a customer, outgoing arcs to charging nodes are disregarded, which is valid thanks to the relaxed battery constraints and the non-profitability of visiting charging nodes. This strengthens the bound in the case where the closest node to a customer node is a charging station since the arc between them would then be the arc with the least cost. The load-based completion bound for a label L_j is presented in Equation (12).

$$
\begin{aligned}
z_L = \min & \sum_{k \in N \setminus S} u_k y_k + u_j \\
\text{s.t. } & T_j^{load} + \sum_{k \in N \setminus S} q_k y_k \leq C \\
& 0 \leq y_k \leq 1, \quad \forall k \in N \setminus S
\end{aligned}
\tag{12}
$$

Another important adjustment compared to the formulation of Righini and Salani [14] is that when selecting the cheapest outgoing arc for a node, *all* arcs to customer nodes are considered, not just arcs to not visited customers, i.e. $\min_{l \in N \setminus R} c_{kl}$ instead of $\min_{l \in N \setminus (S \cup R)} c_{kl}$. Although the second formulation of the two generates a stronger bound, it is more computationally expensive, since the least arc costs must be computed each time the bounding method is used. With our formulation, however, the least cost arc of a node, and further the least time and least energy consumption arc, which will be used in the second bound, can be precomputed.

The time-based bound can be seen as an extended version of a knapsack problem on maximal route time. It considers arc travel time and, to make sure that the battery level stays positive, the possible extra required charging time. To obtain a lower bound, the same assumptions as in the resource bounds are made, i.e. it is assumed that the charging can be done "on the road" and at the fastest charging rate, ρ_1. Most of the notation is the same as in the vehicle load bound, but some new parameters and variables need to be introduced. To handle the extra potential charging time, a new continuous variable ζ is introduced, which is the total extra charging energy that needs to be added in order to ensure battery feasibility. We also introduce the minimum energy consumption r_i of visiting a node $i \in V$, and the minimum travel time t_i to visit a node, $i \in V$. Both are calculated in the same manner as the lowest cost, u_i, by finding the cheapest outgoing arc from a node $i \in V$ for energy and time respectively, i.e. $r_i = \min_{l \in N \setminus R} r_{il}$ and $t_i = \min_{l \in N \setminus R} t_{il}$. The time-based completion bound for a label L_j is presented in Equation (13).

$$\begin{aligned} z_T = \min & \sum_{k \in N \setminus S} u_k y_k + u_j \\ \text{s.t. } & T_j^{\text{time}} + t_j + \zeta/\rho_1 + \sum_{k \in N \setminus S} t_k y_k \leq T_{\max} \\ & r_j + \sum_{k \in N \setminus S} r_k y_k \leq \zeta + T_j^{\text{energy}} \\ & 0 \leq y_k \leq 1, \quad \forall k \in N \setminus S \\ & 0 \leq \zeta \end{aligned} \tag{13}$$

Even if z_T generates a good bound, it is not as straightforward to calculate efficiently as z_L. It can always be solved with an LP-solver, but that would not be that effective. However, if we assume that the problem instance has the properties that $t_1 \geq t_2 \Rightarrow r_1 \geq r_2$, then we can use a greedy algorithm to calculate the bound, presented in Algorithm 2. Note that this assumption holds for the instances tested in this paper, the EVRPTW dataset of Schneider et al. [15], where the relation between travel time and energy consumption is linear, and the service time are same for all customers in one instance. As input to the algorithm we precompute the most profitable customers and add them to a list, denoted K, which is ordered by increasing value of the quotients $\frac{u_k}{t_k + r_k/\rho_1}$. The algorithm then iterates over the nodes in this order and adds the most profitable nodes to the path as long as the max time limit is not violated.

4.2.1 Integration of ng-routes with Completion Bounds

Another popular acceleration strategy for a labeling algorithm is ng-routes, first suggested by [1]. In ng-routes, the elementarity constraints on the paths are partly relaxed allowing revisits to a customer as long as the customer is not in the ng-set of the current node. The ng-set is unique for each node and contains the nodes for which elementarity is checked when extending a label. If a customer is not in the ng-set of the node we are extending the label from, it can be added to the path even if it has been visited before, allowing for subtours. The size of the ng-set, n, is set according to what fits the problem and is often chosen to be the n closest customers to the node, including itself. This is also the approach we have chosen to use. One of the benefits of using ng-routes is that if the best

Algorithm 2 Completion bound with time and battery constraints.

```
$C \leftarrow 0$ // Initial cost
$\sigma \leftarrow \max\{0, (r_j - T^{\text{energy}})/\rho_1\} + t_j$ // Time needed to leave $j$
$T \leftarrow T^{\text{end}} - T_j^{\text{time}} - \sigma$ // Time left
$E \leftarrow \max\{0, T_j^{\text{energy}} - r_j\}$ // Remaining energy
$i \leftarrow 0$
while $T \geq 0$ do
    // Find the most profitable customer to add
    $k \leftarrow K[i]$
    // Check that the customer can be added and have a positive effect
    if $T_j^{cust}[k] = 0 \ \wedge \ u_k < 0$ then
        // Time needed to add customer $k$
        $\sigma \leftarrow \max\{0, (r_k - E)/\rho_1\} + t_k$
        // Check if customer can be fully added, else add partially
        if $T - \sigma \geq 0$ then
            $C \leftarrow C + u_k$
            $T \leftarrow T - \sigma$
            $E \leftarrow \max\{0, E - r_j\}$
        else
            // Percentage that can be added
            $\lambda \leftarrow (\sigma - T)/\sigma$
            $C \leftarrow C + \lambda u_k$
            $T \leftarrow T - \lambda\sigma$
            $E \leftarrow E + \lambda r_k$
        end
    end
    $i$++
end
```

path is elementary when finishing the labeling algorithm, it will be also be an optimal elementary path. Using ng-routes is a powerful acceleration strategy. It can, however, not be used directly with the implemented completion bounds. The reason for this complication is that the implemented completion bounds create *elementary completions* to the partial paths, which are not necessarily optimistic completions to an ng-route. Since subtours are allowed for ng-routes, it has to be considered that a node can be added multiple times to the completion of the path, and the calculation of completion bounds need to be adjusted accordingly.

There are some existing approaches to combine ng-routes with completion bounds in the literature, but none really fits with our problem. For implementations where each problem is solved several times to find elementarity, i.e. iteratively extending the size of the ng-set until the optimal path is elementary, labels from the previous iteration can be used to compute lower bounds on the reduced cost of the completion of a path (used by [13, 4, 12]). Another approach to use completion bounds in the ng-route ESPPRC was suggested by Baldacci et al. [1], and was applied to the electric ng-route SPPRC by Duman et al. [6]. They use the completion bounds as a big part of the solution method starting each solving process by, for each customer, running an exact labeling algorithm with the customer as a start node and with $t^{\text{start}} = T_{\max} - \Delta t$ as the start time. The results from these runs are then used as lower bounds when solving again, but with the start time $t^{\text{start}} = T_{\max} - 2\Delta t$. This is then repeated until the full problem is solved.

However, we want to integrate completion bounds with ng-routes without significant changes to the algorithm. When computing the completion bound we therefore assume that there are no elementary constraints, to ensure it to be an optimistic bound on the solutions also in the case of using ng-routes. However, by considering other constraints, the bound can be strengthened. Firstly a customer cannot be added at all if it is not reachable from the current node, i.e. if $T_j^{\text{time}} + t_{jl} < l_l$. Secondly, given it is reachable, an upper bound on the total number of times a node can be added to the path can be calculated. Provided the least time outgoing arc $t_{lk}^{\min}$ from node l and the least time incoming arc $t_{kl}^{\min}$ to node l, then $t_{lk}^{\min} + t_{kl}^{\min}$ is the minimum time required to leave and come back to node l. Furthermore, the earliest arrival at the node l can be calculated as $T_l^{\text{start}} = \max\{T_j^{\text{time}} + t_{jl}, e_l\}$, comparing the earliest allowed arrival e_l with the possible earliest arrival given the label T_j^{time} at node j and the travel time t_{jl}. Using both of these, an upper bound on the number of visits at node l can be calculated by $\left\lfloor \frac{l_l - T_l^{\text{start}}}{t_{lk}^{\min} + t_{kl}^{\min}} + 1 \right\rfloor$.

To further strengthen the upper bound on the number of visits, the elementary constraints that exist in an ng-routes setting can be taken into consideration as follows. In order to be allowed to come back to a node l that already has been visited, the route must be extended from a node k for which l is not in its ng-set. Provided the least time outgoing arc from node l to a node for which l is not in its ng-set $t_{lk}^{\min_{\text{ng}}}$ and the least time incoming arc from a node for which l is not in its ng-set to node l, $t_{kl}^{\min_{\text{ng}}}$, then $t_{lk}^{\min_{\text{ng}}} + t_{kl}^{\min_{\text{ng}}}$ can be used instead as the minimum time required to leave and come back to node l. The strengthened upper bound on the number of visits at node l can hence be calculated as

$$\left\lfloor \frac{l_l - T_l^{\text{start}}}{t_{lk}^{\min_{\text{ng}}} + t_{kl}^{\min_{\text{ng}}}} + 1 \right\rfloor.$$

In practice this adaption to make completion bounds work with ng-routes is implemented by adding nodes to the path in order of decreasing profitability given that it is reachable and that the multiplicity is not violated. Note that in order to ensure an optimistic bound, given the ng-setting, a node can be added multiple times in a row as long as the multiplicity constraint is not violated and that there is place left in the knapsack.

5 Experimental Results

The performance of the labeling algorithm with the suggested acceleration strategies is evaluated using the EVRPTW benchmark data set of Schneider et al. [15] which is based on Solomon's benchmark instances for the VRPTW [16]. Tests are performed on all 29 instances from the groups C100, RC100, and R100, that have 100 customer nodes with narrow time windows and 20 charging nodes with no time windows. Each instance is tested in three different simulated column generation environments, resulting in a total of $3 \times 27 = 89$ instances.

To simulate a column generation environment, values of the dual variables for customers, $\pi_i,\ i \in N$, are generated as random integer variables from a uniform distribution on the interval $\{0, \ldots, 20\}$ as suggested by Feillet et al. [8]. These are used to update the arc costs of outgoing arcs from customers nodes $i \in N$: $c_{ij} = d_{ij} - \pi_i$, where d_{ij} is the Euclidean distance between node i and node j. For all other nodes $i \in V \setminus N$ the outgoing arc costs are set to $c_{ij} = d_{ij}$. The generated dual values are available in the public repo (available at `https://gitlab.liu.se/eliro15/labeling-algorithm-for-evrptw-plr`).

Table 1 Number of instances solved for each group and acceleration strategy

Series	basic	ng-routes	bounds load	bounds time	ng-routes+bounds
$C100$	21/27	27/27 (20)	21/27	21/27	27/27 (20)
$RC100$	21/24	24/24 (19)	23/24	24/24	24/24 (19)
$R100$	18/36	25/36 (16)	20/36	21/36	29/36 (17)
Total	60/87	76/87 (55)	64/87	66/87	80/87 (56)

Table 2 Aggregated results for each group and acceleration strategy

Series	basic	ng-routes		bounds load		bounds time		ng-routes+bounds	
	$\bar{t}$	$\bar{t}$	$\Delta t[\%]$	$\bar{t}$	$\Delta t[\%]$	$\bar{t}$	$\Delta t[\%]$	$\bar{t}$	$\Delta t[\%]$
$C100$	15.11	3.09	-79.60	9.06	-40.07	1.80	-88.06	**0.74**	**-95.11**
$RC100$	24.33	12.69	-47.87	14.15	-41.84	11.83	-51.38	**6.17**	**-74.64**
$R100$	620.67	515.09	-17.01	417.23	-32.79	256.75	-58.63	**165.01**	**-73.41**

In the dataset for EVRPTW it is assumed that the charging curve is linear, where the battery capacity Q and the inverse recharging rate ω is unique for each instance. To adapt the dataset to a piecewise linear charging function, we fit the charging curve in Figure 1, by scaling the time of breakpoints by $\omega Q \frac{t_{\text{old}}}{1.01}$ and the battery levels by $Q\frac{e_{\text{old}}}{16}$. From this it is then easy to calculate, for each instance, the set of pieces in the piecewise linear charging function with its corresponding breakpoints and recharging rates which are available in our public repo.

Before running the algorithm the graph is reduced by removing infeasible arcs, i.e. arcs between nodes $i \in V$ and $j \in V$ such that $e_i + t_{ij} > l_j$, $q_i + q_j > C$ or $r_{ij} > Q$.

The algorithm was implemented in C++ and compiled with GCC 11.4.0. It was tested on a PC with AMD Ryzen Pro565OU CPU at 2.301 GHz, 32 GB RAM on Ubuntu with WSL.

We report results of each instance solved using (i) no acceleration strategy, (ii) ng-routes, (iii) resource bounds + load-based completion bound, (iv) resource bounds + time-based completion bound. Finally, a combination of all acceleration strategies is tested (v): ng-routes + resource bounds + load-based completion bound + time-based completion bound. A maximum time of 1 hour was set for each instance.

To minimize unnecessary computational effort, the completion bounds are only computed if the resource consumption for the constrained resource exceeds 20% of the total resource availability. This threshold was chosen from pretrial tests (Figure 2 in Appendix A) of different thresholds on resource consumption. In the combined setting, the time-based bound is computed first and if $T_j^{\text{cost}} + z_T > UB$ the label is discarded straight away. Otherwise, the load-based bound is computed to check if $T_j^{\text{cost}} + z_L > UB$.

The size of the ng-sets were set to 10 ($C100$), 15 ($RC100$), and 20 ($R100$) nodes respectively. For each group of instances, these were the smallest sizes of multiples of 5 of the ng-sets for which the optimal solutions of at least half of the instances were elementary.

In Table 1, we report how many instances from each group were solved to optimality within the 1-hour time limit. For the acceleration strategies using ng-routes, it is reported in parentheses how many of the optimal paths were elementary. It is worth to remember that in a column generation environment, as long as there is one elementary path with negative reduced cost among the finished paths, the result is useful. Full results for each instance and acceleration strategy can be found in Table 3–Table 5 in Appendix B.

Table 2 shows aggregated time results per group and acceleration strategy. To make a fair comparison, only instances that were solved with an elementary optimal solution by all acceleration strategies within the 1-hour limit are included in the calculation. Deviation for a strategy x and group y is calculated compared to the basic version by $\Delta\bar{t}^y_x = 100 \cdot (\bar{t}^y_x - \bar{t}^y_{\text{basic}})/\bar{t}^y_{\text{basic}}$.

Comparing the results in Table 1 and Table 2, we can state that the combination of ng-routes and completion bounds (*v*) is the most effective acceleration strategy. It solves the largest number of instances within the time limit, and achieves the largest reduction in computational time compared to the basic version, from −73% to −95% in average, depending on the instance group. When it comes to the computational time, the pattern for the instances not included in the aggregated results is the same as for the instances included, i.e. the combination of ng-routes and completion bounds is the most effective strategy.

Comparing the two implemented completion bounds, the time-based bound (*iv*) is on average more efficient than the load-based bound (*iii*) for all groups of instances, and for some the difference is significant. This indicates that the vehicle load is not as often a binding constraint as the time constraint for the instances tested, at least when taking charging time into account.

Looking a bit closer at the full results, Table 3–Table 5 in Appendix B, it can be seen that the main difference in computational time between the other acceleration strategies and acceleration strategy (*v*) shows on the more difficult instances. On instances that are easier to solve, the difference in computational time between the strategies is not as large, and often one of the plain bounding strategies is the most effective. The reason for this is most likely that completion bounds generated when combining it with ng-routes is weaker compared to when not combined with ng-routes, and the usage of ng-routes does not compensate for this on easier instances.

6 Conclusion

In this paper, we propose bounding methods to accelerate a labelling algorithm used for solving the ESPPRC as the pricing problem of the EVRPTW with a piecewise linear charging curve and partial recharging. Two types of bounding methods are implemented, resource bounds and completion bounds. For completion bounds we implemented two versions, one load based and one time based. The latter is a new stronger completion bound that takes travel time as well as charging time into account. A strength of both the completion bounds is that they are cheap to compute, since the information required can be preprocessed. Furthermore, we propose a way to integrate the bounding methods with ng-routes, another popular acceleration strategy.

Experimental results show that the combination of ng-routes and bounding methods was most efficient, reducing the computational time from −73% to −95% in average, depending on the instance group, compared to a basic labelling algorithm on benchmark instances with 100 customers and 20 chargers. The results also show that the time-based completion bound was more efficient than the load-based completion bound for all groups of instances.

Our efficient way of computing the time-based completion bound uses a specific relation between travel time and energy consumption. Of interest for future work is to investigate efficient ways of computing this time-based completion bound when these assumptions do not hold.

References

1 Roberto Baldacci, Aristide Mingozzi, and Roberto Roberti. New route relaxation and pricing strategies for the vehicle routing problem. *Oper. Res.*, 59(5):1269–1283, 2011. doi:10.1287/OPRE.1110.0975.

2 Moritz Baum, Julian Dibbelt, Andreas Gemsa, Dorothea Wagner, and Tobias Zündorf. Shortest feasible paths with charging stops for battery electric vehicles. *CoRR*, abs/1910.09812, 2019. arXiv:1910.09812.

3 Natashia Boland, John Dethridge, and Irina Dumitrescu. Accelerated label setting algorithms for the elementary resource constrained shortest path problem. *Oper. Res. Lett.*, 34(1):58–68, 2006. doi:10.1016/J.ORL.2004.11.011.

4 Claudio Contardo and Rafael Martinelli. A new exact algorithm for the multi-depot vehicle routing problem under capacity and route length constraints. *Discret. Optim.*, 12:129–146, 2014. doi:10.1016/J.DISOPT.2014.03.001.

5 Guy Desaulniers, Fausto Errico, Stefan Irnich, and Michael Schneider. Exact algorithms for electric vehicle-routing problems with time windows. *Oper. Res.*, 64(6):1388–1405, 2016. doi:10.1287/OPRE.2016.1535.

6 Ece Naz Duman, Duygu Tas, and Bülent Çatay. Branch-and-price-and-cut methods for the electric vehicle routing problem with time windows. *Int. J. Prod. Res.*, 60(17):5332–5353, 2022. doi:10.1080/00207543.2021.1955995.

7 Jenny Enerbäck. A labelling algorithm for the resource constrained elementary shortest path problem. *Master Thesis, Linköping University*, 2024. URL: urn:nbn:se:liu:diva-205286.

8 Dominique Feillet, Pierre Dejax, Michel Gendreau, and Cyrille Gueguen. An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks*, 44(3):216–229, 2004. doi:10.1002/NET.20033.

9 Stefan Irnich and Guy Desaulniers. Shortest path problems with resource constraints. In *Column generation*, pages 33–65. Springer, 2005.

10 Edward Lam, Guy Desaulniers, and Peter J. Stuckey. Branch-and-cut-and-price for the electric vehicle routing problem with time windows, piecewise-linear recharging and capacitated recharging stations. *Comput. Oper. Res.*, 145:105870, 2022. doi:10.1016/J.COR.2022.105870.

11 Alejandro Montoya, Christelle Guéret, Jorge E. Mendoza, and Juan G. Villegas. The electric vehicle routing problem with nonlinear charging function. *Transportation Research Part B: Methodological*, 103:87–110, 2017. doi:10.1016/j.trb.2017.02.004.

12 Diego Pecin, Artur Alves Pessoa, Marcus Poggi, and Eduardo Uchoa. Improved branch-cut-and-price for capacitated vehicle routing. *Math. Program. Comput.*, 9(1):61–100, 2017. doi:10.1007/S12532-016-0108-8.

13 Rafael Martinelli Pinto. Exact algorithms for arc and node routing problems. *These de doctorat, Pontifcia Universidade Católica do Rio de Janeiro*, 2012.

14 Giovanni Righini and Matteo Salani. Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. *Discret. Optim.*, 3(3):255–273, 2006. doi:10.1016/J.DISOPT.2006.05.007.

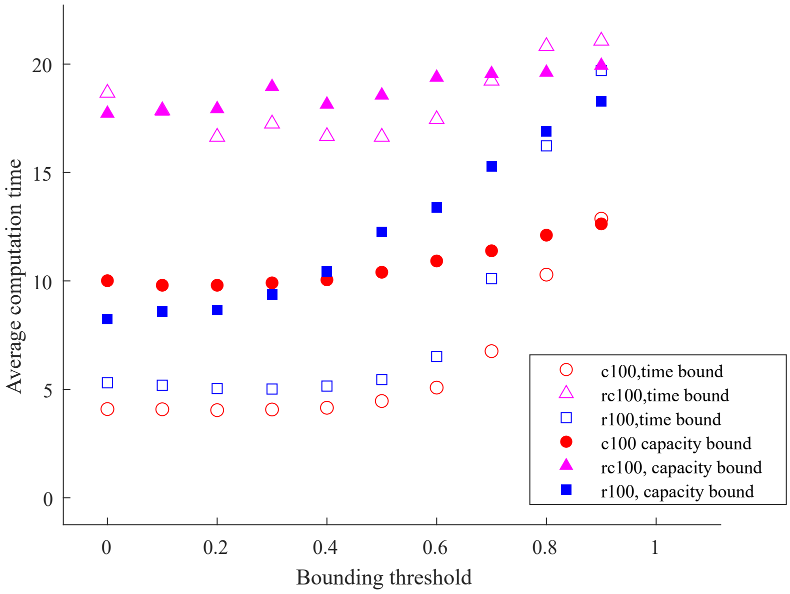
15 Michael Schneider, Andreas Stenger, and Dominik Goeke. The electric vehicle-routing problem with time windows and recharging stations. *Transp. Sci.*, 48(4):500–520, 2014. doi:10.1287/TRSC.2013.0490.

16 Marius M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Oper. Res.*, 35(2):254–265, 1987. doi:10.1287/OPRE.35.2.254.

A Pretrial results

In Figure 2 we present the average computation time of instances in $C100$ ($c101$, $c105$, $c106$, $c107$, $c108$), $RC100$ ($rc101$, $rc102$, $rc105$, $rc106$, $rc107$) and $R100$ ($r101$, $r105$, $r109$) using different starting thresholds of resource consumption for the two completion bounds.

Figure 2 Average computation times using different bounding thresholds.



B Results

We report results of each instance in Table 3–Table 5, divided by the three simulated column generation environments, solved using (*i*) no acceleration strategy, (*ii*) ng-routes, (*iii*) resource bounds + load-based completion bound, (*iv*) resource bounds + time-based completion bound and (*v*): ng-routes + resource bounds + load-based completion bound + time-based completion bound. For each instance and strategy we report the solving time in seconds, and how many non-dominated labels were left at the end of the algorithm, in thousands. We indicate for the results using ng-routes with a star which instances did not have an elementary optimal solution. The ng-set size was set to 10, 15, and 20 for the instance groups $C100$, $RC100$, and $R100$, respectively. The instances that could not be solved within the 1-hour time limit are indicated with '-'.

Table 3 C100, RC100 and R100: basic version, ng-routes, completion bounds and ng-routes with bounds. First simulated column generation environment.

	basic		ng-routes		bound load		bound time		ng+bounds	
	Labels	Time	Labels	Time	Labels	Time	Labels	Time	Labels	Time
*c*101	10.57	0.72	13.64	1.06	7.02	0.38	**6.64**	**0.26**	6.91	0.33
*c*102	292.68	1639.79	61.19*	20.03*	257.23	828.44	**153.99**	**156.62**	41.22*	7.04*
*c*103	-	-	187.81	450.61	-	-	-	-	**122.50**	**81.83**
*c*104	-	-	300.45*	1136.58*	-	-	-	-	237.44*	509.01*
*c*105	13.75	1.22	16.81	1.49	11.00	0.76	**7.28**	**0.38**	8.65	0.42
*c*106	22.98	2.72	25.80	2.97	19.07	1.65	8.87	0.48	**9.04**	**0.39**
*c*107	16.64	1.76	19.37	2.21	14.03	1.20	**8.73**	**0.54**	10.00	0.57
*c*108	32.11	5.03	34.28	4.65	27.84	3.05	15.90	1.01	**15.64**	**0.85**
*c*109	70.44	21.13	58.71	12.37	63.35	11.06	39.55	3.57	**29.65**	**2.34**
*rc*101	9.92	0.23	16.29	0.70	6.19	0.15	**5.42**	**0.13**	7.46	0.19
*rc*102	56.39	4.33	41.89*	2.77*	48.15	2.75	**46.68**	**2.56**	31.40*	1.33*
*rc*103	331.06	182.51	124.35	24.99	277.25	94.08	257.62	74.68	**100.38**	**10.83**
*rc*104	-	-	232.14	93.65	-	-	1013.24	2355.03	**182.59**	**34.67**
*rc*105	32.25	1.67	42.00	3.69	26.43	1.08	**25.69**	**1.03**	29.92	1.34
*rc*106	26.83	1.08	38.22	2.91	22.44	0.76	**21.59**	**0.71**	26.95	1.02
*rc*107	128.75	18.88	127.96	21.16	114.46	12.95	113.12	10.91	**107.04**	**10.32**
*rc*108	284.19	100.13	223.03	72.62	254.82	62.64	250.77	52.10	**190.37**	**33.86**
*r*101	9.96	0.24	30.10	2.85	4.08	0.08	**1.86**	**0.05**	2.66	0.07
*r*102	268.04	455.46	192.05*	676.26*	187.85	152.03	**165.44**	**105.60**	89.49*	76.10*
*r*103	-	-	580.34*	3383.36*	-	-	-	-	375.53*	847.43*
*r*104	-	-	-	-	-	-	-	-	-	-
*r*105	32.35	2.15	79.55	26.84	13.14	0.50	**8.39**	**0.32**	10.85	0.51
*r*106	361.49	1957.72	258.61*	1365.62*	252.39	659.79	**221.34**	**563.77**	127.36*	171.22*
*r*107	-	-	-	-	-	-	-	-	250.37*	1364.76*
*r*108	-	-	-	-	-	-	-	-	-	-
*r*109	77.27	11.18	124.72	57.29	45.36	3.40	**28.16**	**2.07**	35.72	3.38
*r*110	609.45	1034.56	454.91	773.58	528.69	620.33	388.87	423.37	**284.19**	**241.75**
*r*111	617.17	1218.69	408.33	996.71	538.91	709.02	398.09	509.14	**268.08**	**323.70**
*r*112	-	-	-	-	-	-	-	-	**788.28**	**3323.50**

Table 4 C100, RC100 and R100: basic version, ng-routes, completion bounds and ng-routes with bounds. Second simulated column generation environment.

	basic		ng-routes		bound load		bound time		ng+bounds	
	Labels	Time	Labels	Time	Labels	Time	Labels	Time	Labels	Time
*c*101	6.26	0.30	8.07	0.30	4.15	0.17	**3.04**	**0.10**	3.60	0.12
*c*102	262.03	993.25	41.29*	5.61*	232.83	516.59	**129.38**	**64.23**	22.86*	2.10*
*c*103	-	-	129.01*	82.20*	-	-	-	-	96.60*	26.76*
*c*104	-	-	133.37*	229.87*	-	-	-	-	104.75*	80.61*
*c*105	9.80	0.58	12.32	0.67	7.69	0.41	**5.69**	**0.22**	6.75	0.25
*c*106	16.49	1.41	18.35	1.31	13.99	0.91	**8.88**	**0.38**	10.74	0.44
*c*107	10.33	0.78	12.41	0.81	8.78	0.58	**6.56**	**0.32**	7.36	0.34
*c*108	23.15	2.71	24.61	2.18	20.09	1.70	**13.79**	**0.79**	15.04	0.78
*c*109	46.66	10.22	36.66	5.27	43.07	6.25	32.68	2.81	**26.32**	**2.00**
*rc*101	9.22	0.25	14.67	0.63	4.92	0.13	**4.04**	**0.12**	5.29	0.16
*rc*102	63.35	5.80	50.94	4.69	42.70	2.34	39.28	1.96	**33.93**	**2.67**
*rc*103	233.18	97.14	135.98*	28.09*	**179.02**	**42.10**	175.89	43.30	105.43*	13.47*
*rc*104	-	-	231.74	85.70	1254.22	2238.00	1077.48	1492.99	**185.34**	**37.29**
*rc*105	23.85	0.89	33.49	1.98	18.26	0.54	**17.52**	**0.53**	21.51	0.73
*rc*106	21.40	0.81	31.36	1.76	16.33	0.51	**15.46**	**0.51**	19.52	0.68
*rc*107	78.47	8.19	88.11	11.74	68.98	5.24	**67.85**	**4.96**	71.19	5.23
*rc*108	161.45	39.18	135.66	24.69	140.67	19.64	135.28	19.30	**107.70**	**10.69**
*r*101	5.49	0.11	14.35	0.65	3.24	0.07	**1.94**	**0.05**	2.73	0.07
*r*102	509.03	1610.95	264.09	672.55	415.35	873.40	341.72	547.88	**177.42**	**201.77**
*r*103	-	-	388.42*	1165.05*	-	-	-	-	219.83*	232.07*
*r*104	-	-	-	-	-	-	-	-	-	-
*r*105	15.31	0.56	36.57	4.35	8.11	0.24	**6.22**	**0.19**	8.18	0.29
*r*106	-	-	411.79*	1755.61*	-	-	714.61	2795.80	289.07*	652.41*
*r*107	-	-	519.23*	2438.46*	-	-	-	-	304.52*	571.81*
*r*108	-	-	-	-	-	-	-	-	-	-
*r*109	64.95	13.62	107.91	72.88	48.34	5.45	**41.61**	**4.12**	52.29	6.84
*r*110	379.97	612.04	335.82	628.84	332.98	317.98	290.90	200.27	**237.60**	**146.19**
*r*111	415.15	640.51	339.85	546.87	370.74	274.64	332.89	270.65	**250.83**	**178.28**
*r*112	-	-	808.89*	3512.37*	-	-	-	-	645.10*	1889.91*

Table 5 C100, RC100 and R100: basic version, ng-routes, completion bounds and ng-routes with bounds. Third simulated column generation environment.

	basic		ng-routes		bound load		bound time		ng+bounds	
	Labels	Time	Labels	Time	Labels	Time	Labels	Time	Labels	Time
*c*101	10.74	0.56	14.58	0.68	6.35	0.25	**5.26**	**0.20**	6.28	0.25
*c*102	189.41	212.56	43.98	4.73	158.19	126.25	89.23	18.95	**23.85**	**1.50**
*c*103	-	-	145.53*	120.09*	-	-	-	-	88.56*	37.24*
*c*104	-	-	250.34*	347.26*	-	-	-	-	177.54*	130.94*
*c*105	14.92	0.81	19.44	1.11	11.21	0.53	**7.25**	**0.28**	8.65	0.31
*c*106	24.54	1.81	28.38	2.02	20.27	1.25	10.23	0.42	**9.86**	**0.41**
*c*107	16.19	1.03	20.00	1.29	13.48	0.75	**9.10**	**0.39**	10.46	0.50
*c*108	33.56	4.07	35.73	3.97	28.20	2.55	15.52	0.84	**13.28**	**0.58**
*c*109	67.46	17.71	54.62	9.49	59.27	12.37	36.37	3.24	**22.81**	**1.65**
*rc*101	8.72	0.18	14.43	0.50	5.70	0.12	**5.05**	**0.12**	7.06	0.19
*rc*102	81.43	10.20	42.28*	2.91*	**69.06**	**6.66**	65.75	6.87	31.21*	1.61*
*rc*103	277.02	170.70	94.47*	11.93*	228.45	118.19	**213.85**	**102.08**	69.46*	5.94*
*rc*104	-	-	202.47*	54.83*	1212.49	2985.51	**1078.13**	**2329.54**	169.58*	32.54*
*rc*105	26.29	1.04	34.30	2.21	20.42	0.70	**19.25**	**0.68**	23.30	0.94
*rc*106	23.22	0.75	31.87	1.56	17.90	0.51	**16.92**	**0.51**	21.42	0.72
*rc*107	105.39	10.72	110.37	13.92	90.57	8.16	**87.71**	**7.46**	88.95	8.26
*rc*108	198.30	41.37	163.56	25.93	175.65	31.02	169.05	25.43	**135.59**	**18.03**
*r*101	8.48	0.28	22.52	2.25	3.79	0.09	**1.50**	**0.05**	2.01	0.07
*r*102	-	-	235.24*	1178.80*	291.00	1939.15	**193.89**	**891.05**	117.96*	266.17*
*r*103	-	-	-	-	-	-	-	-	343.92*	3242.89*
*r*104	-	-	-	-	-	-	-	-	-	-
*r*105	26.81	1.96	59.52	15.06	12.38	0.61	**5.68**	**0.29**	7.25	0.45
*r*106	-	-	278.71*	1272.39*	332.47	2962.30	**217.65**	**1046.66**	134.17*	244.67*
*r*107	-	-	-	-	-	-	-	-	284.58*	2133.11*
*r*108	-	-	-	-	-	-	-	-	-	-
*r*109	82.11	32.52	129.89	196.64	51.20	11.44	**26.56**	**4.59**	36.29	9.02
*r*110	653.41	1482.01	456.48	1007.39	563.37	1211.10	400.06	791.16	**260.14**	**332.05**
*r*111	848.99	3269.41	615.19	3239.52	730.46	2548.20	481.84	1353.81	**376.60**	**1195.76**
*r*112	-	-	-	-	-	-	-	-	-	-

Periodic Event Scheduling with Flexible Infrastructure Assignment

Enrico Bortoletto
Zuse Institute Berlin, Germany

Rolf Nelson van Lieshout
Eindhoven University of Technology, The Netherlands

Berenike Masing
Zuse Institute Berlin, Germany

Niels Lindner
Zuse Institute Berlin, Germany

Abstract

We present novel extensions of the Periodic Event Scheduling Problem (PESP) that integrate the assignment of activities to infrastructure elements. An application of this is railway timetabling, as station and platform capacities are limited and need to be taken into account. We show that an assignment of activities to platforms can always be made periodic, and that it can be beneficial to allow larger periods for the assignment than for the timetable. We present mixed-integer programming formulations for the general problem, as well as for the practically relevant case when multiple platforms can be considered equivalent, for which we present a bipartite matching approach. We finally test and compare these models on real-world instances.

2012 ACM Subject Classification Mathematics of computing → Integer programming; Applied computing → Transportation; Mathematics of computing → Combinatorial optimization; Mathematics of computing → Matchings and factors

Keywords and phrases Periodic Event Scheduling, Periodic Timetabling, Railway Timetabling, Matchings, Infrastructure Assignments, Platform Assignments, Station Capacities

Digital Object Identifier 10.4230/OASIcs.ATMOS.2024.4

Funding *Enrico Bortoletto*: Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – The Berlin Mathematics Research Center MATH+ (EXC-2046/1, project ID: 390685689).
Berenike Masing: Funded within the Research Campus MODAL, funded by the German Federal Ministry of Education and Research (BMBF) (fund number 05M20ZBM).

Acknowledgements We thank DB InfraGO AG for providing data for the S-Bahn Berlin network.

1 Introduction

Out of the many interacting pieces of public transportation services, a key determinant in the puzzle is the timetable. This is an omnipresent yet flexible concern in the operators' minds [3], as well as one of the data of most interest to passengers [14]. These are but two of the reasons for which timetabling and in particular periodic timetabling has received substantial attention in the past. The modeling framework of the Periodic Event Scheduling Problem (PESP) was initially formulated by Serafini and Ukovich [16], quickly found to have rich and interesting underlying structures [1, 9, 12, 13], and studied ever since, also in integration with various other problems, such as [5, 10, 11, 15, 17].

Of special interest in this work is the question of solving PESP while making sure that the produced timetable accounts for various infrastructural constraints of high practical interest in railway operations, involving safety of operations and physical occupation of

24th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2024).
Editors: Paul C. Bouman and Spyros C. Kontogiannis; Article No. 4; pp. 4:1–4:18
OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

tracks. In particular, we focus on what is called *track occupation problem* in [11], which entails ensuring that no two vehicles are ever scheduled to occupy the same point in time and space. This is of particular interest, for example, when planning dwelling activities of trains at the same platform. The need to respect a given association of the activities to be scheduled to infrastructure elements led to *Infrastructure-Aware* PESP (IPESP), which can be formulated as a mixed-integer linear program, by using well-known PESP constraints as foundation [2, 11].

We present two main contributions: At first, we generalize IPESP to *Infrastructure-Aware* PESP *with Assignment* (IPESPA), by integrating the assignment of activities to infrastructure elements into the periodic timetabling problem. We show that such an assignment can always be made periodic, without impact on the timetable, and highlight how it can be advantageous for the period of such an infrastructure assignment to be larger than the one of the timetable, with an elucidatory example. We then also present a mixed-integer programming formulation for IPESPA, that generalizes the so-called Q0-constraints of [11].

The setting above is for the general case, in which we allow any map of activities to sets of infrastructure elements. In a second step, we then restrict our inquiry to a more common and practical use-case, where multiple infrastructure elements can be considered as equivalent. We then assume that activities can only be assigned to one element, but this element is allowed to have a capacity larger than one. For example, the two sides of a platform are oftentimes equivalent options to choose, and we might consider such a platform as an element with capacity two. The main achievement is a mixed-integer programming formulation for this *Infrastructure-Aware* PESP *with Capacities* (IPESPC), a model much more compact than the IPESPA model, based on matchings in certain auxiliary bipartite graphs. Finally, this allows us also to derive two new alternative mixed-integer programming formulations for standard IPESP, i.e., IPESPC with unit capacities, beyond those of [2, 11].

We compare our new formulations on three realistic instances, both in the case of unit and larger capacities, and demonstrate their computational feasibility and practical benefit.

Section 2 recalls the Periodic Event Scheduling Problem and its infrastructure-aware extension IPESP. Section 3 introduces IPESPA, discusses the theory of general infrastructure assignments, and presents a mixed-integer programming formulation. In Section 4, we restrict to IPESPC and present a matching-based MIP model, along with the resulting new formulations for IPESP. We evaluate the computational power of our models in Section 5, before concluding the paper in Section 6.

The present work is a direct consequence of the fruitful connections and conversations that were had during ATMOS 2023 [4], and we thereby thank the organizing committee for fostering the transport optimization community.

2 Periodic Event Scheduling and Infrastructure Awareness

The Periodic Event Scheduling Problem (PESP) is the standard model to compute and optimize timetables for public transport. It is formulated as follows.

▶ **Definition 1** ([16]). *Consider a directed graph G with vertex set $V(G)$ and arc set $A(G)$, together with $T \in \mathbb{N}$, vectors $\ell, u \in \mathbb{R}^{A(G)}$, and $w \in \mathbb{R}^{A(G)}_{\geq 0}$. The* Periodic Event Scheduling Problem (PESP) *is to find vectors $\pi \in \mathbb{R}^{V(G)}$ and $x \in \mathbb{R}^{A(G)}$ such that*

a) $\pi_j - \pi_i \equiv x_a \bmod T$ *for all* $a = (i,j) \in A(G)$,

b) $\ell \leq x \leq u$,

c) $w^\top x$ *is minimum,*

or to decide that no such π and x exist.

In public transportation practice the directed graph G is oftentimes a so called *event-activity network*. There nodes $V(G)$ are the events, typically arrival or departure events, whereas arcs $A(G)$ are the activities, typically driving from a departure to an arrival, or dwelling from an arrival to a departure. The number $T \in \mathbb{N}$ is the *period time*, and determines after how long each event should repeat. Then the vectors π and x sought by PESP are called *periodic timetable* and *periodic tension*, respectively, where the former represents T-periodic timestamps denoting at which point of each period each event should occur, and the latter instead denotes the duration of the activities in-between events. PESP instances are commonly denoted as (G, T, ℓ, u, w).

Note that the simultaneous use of timetable and tension variables is primarily for ease of expression, since one can always be recovered from the other. In fact, given a periodic timetable π, a corresponding tension is quickly found by setting $x_a := [\pi_j - \pi_i - \ell_a]_T + \ell_a$ for every $a = (i, j) \in A(G)$, where $[\cdot]_T$ denotes the modulo T operator with values in $[0, T)$. Likewise, given a periodic tension, a corresponding timetable is quickly found by a connected graph traversal [7, Theorem 9.8].

For in-depth analysis of the many properties of PESP, we refer to the literature, starting with [8]. Multiple mixed-integer program formulations for PESP are known [7]. For simplicity, here we choose the *standard formulation* [16], which models PESP by linearizing the modulo constraints by use of auxiliary integer variables p_{ij}, called periodic offsets:

$$\min \sum_{(i,j)\in A(G)} w_{ij} x_{ij} \tag{1a}$$

$$\text{s.t.} \quad \pi_j - \pi_i + T p_{ij} = x_{ij} \quad \forall (i,j) \in A(G), \tag{1b}$$

$$0 \le \pi_i < T \quad \forall i \in V(G), \tag{1c}$$

$$\ell_{ij} \le x_{ij} \le u_{ij} \quad \forall (i,j) \in A(G), \tag{1d}$$

$$p_{ij} \in \mathbb{Z} \quad \forall (i,j) \in A(G). \tag{1e}$$

Now we continue with the basic extension of PESP with infrastructure awareness, as per [2]. First of all we define an *infrastructure map* $\eta \colon \mathcal{A} \to E$, mapping certain arcs $\mathcal{A} \subseteq A(G)$ to a set of *infrastructure elements* E. This map encodes an assignment of activities to infrastructure, implying where those activities will physically take place within the network. We denote $\mathcal{A}_e := \eta^{-1}(e)$, for $e \in E$. For each infrastructure element we also have a minimum headway time $h \in \mathbb{R}^E_{\ge 0}$, indicating how long this element needs to be unoccupied between uses of different vehicles. As in [2], we assume that for each $e \in E$ either $h_e > 0$ or $\ell_a > 0$ for all $a \in \mathcal{A}_e$, to avoid pathological cases.

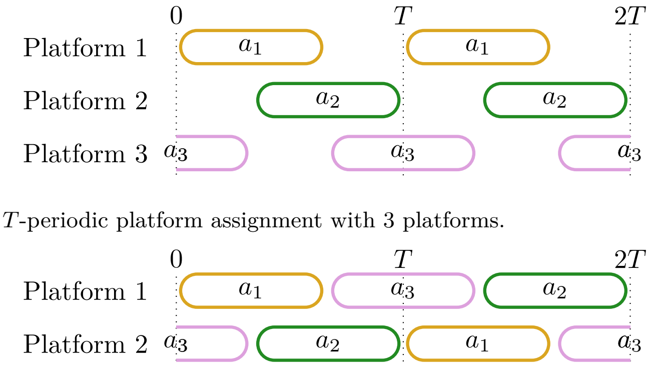
Considering then arcs $a_1 = (i_1, j_1)$ and $a_2 = (i_2, j_2)$ in $\mathcal{A}$, and such that $a_1 \neq a_2$ and $\eta(a_1) = \eta(a_2)$, we say that a_1 does not *h-overlap* a_2 if it holds that

$$[\pi_{i_2} - \pi_{i_1}]_T \ge x_{a_1} + h_e. \tag{2}$$

The constraint (2) is called Q0-constraint in [11], but there is also another possible equivalent formulation, namely the Q4-constraint (“butterfly constraint”). These entail, for each pair of arcs a_1 and a_2 as above, the addition of auxiliary arcs (j_1, i_2) and (j_2, i_1) with lower bound h_e and upper bound $T - h_e$, and then imposing total tension exactly equal to T along the 4-cycle $q(a_1, a_2) := (i_1, j_1, i_2, j_2, i_1)$. That is,

$$\sum_{a \in q(a_1,a_2)} x_a = T. \tag{3}$$

Then (3) holds for $q(a_1, a_2)$ if and only if a_1 and a_2 do not h-overlap each other [11].



(a) T-periodic platform assignment with 3 platforms.

(b) $2T$-periodic platform assignment with 2 platforms.

Figure 1 Two platform assignments of the same three activities.

We say that a set $S \subseteq \mathcal{A}$ is *h-conflict-free* if no arc in S h-overlaps another, and furthermore $x_a + h_e \leq T$ for all $e \in E$ and all $a \in S \cap \mathcal{A}_e$.

▶ **Definition 2** ([2]). *Let (G, T, ℓ, u, w) be a* PESP *instance, let $\eta \colon \mathcal{A} \to E$ be an infrastructure map, and let $h \in \mathbb{R}^E_{\geq 0}$. The* Infrastructure-Aware PESP (IPESP) *is to find a solution (π, x) to* PESP *on (G, T, ℓ, u, w) such that $\mathcal{A}$ is h-conflict-free and the solution is optimal, or to decide that no such solution exists.*

The PESP mixed-integer program (1), together with either all necessary Q0-constraints (2) or Q4-constraints (3), solves Infrastructure-Aware PESP.

This extended form of PESP implicitly assumes two rather strict properties. Firstly, the map η is, by definition, mapping each arc in $\mathcal{A}$ to a single $e \in E$, thereby presuming that such an infrastructure assignment has already been fixed. This implies that every activity must repeat every period always on the same infrastructure. However common, this need not be the case, and as we will see in the next section, it shall not.

3 General Infrastructure Awareness with Flexible Infrastructure Maps

Let us begin by considering the following illustrative example, motivating our work.

▶ **Example 3.** Consider an IPESP situation where we have $T = 30$ minutes, and three dwelling activities $a_k = (i_k, j_k)$ for $k \in \{1, 2, 3\}$. Suppose further that there are three platforms e_1, e_2, e_3 with $\eta(a_k) = e_k$ for all $k \in \{1, 2, 3\}$, without headway requirements. Let $\pi_{i_1} = 0$, $\pi_{i_2} = 10$, $\pi_{i_3} = 20$, and $\pi_{j_1} = 20$, $\pi_{j_2} = 0$, $\pi_{j_3} = 10$, meaning that each dwelling activity is scheduled for 20 min. In Figure 1a, we see how this would play out. Crucially, this leaves each platform unoccupied for 10 minutes per period of 30 minutes.

Suppose we were now to allow all activities to be assigned to either platform, and possibly to different platforms at different periodic repetitions. Then, the configuration of Figure 1b would be possible. With the same timetable as before, only two platforms are required now. This enables a more efficient use of the existing infrastructure.

Moreover, IPESP always assumes that no infrastructure element can be occupied for longer than T. This might however be practically necessary, e.g., due to regulations on minimum turnaround times.

3.1 Problem Definition and Periodizability of Assignments

We now present the tools to do timetabling while ensuring efficient use of the underlying infrastructure. Consider the flexible infrastructure map $\eta\colon \mathcal{A} \to \mathcal{E} \subseteq 2^E$, and so having the option to choose where to have activities occur. Given a periodic timetable π and tension x on a PESP instance (G, T, ℓ, u, w), we call

$$\mathcal{I} := \left\{ I_a^{(k)} \;\middle|\; a \in \mathcal{A}, k \in \mathbb{Z} \right\} \tag{4}$$

the *realisation* of (π, x), where $I_a^{(k)} := [\pi_i + kT, \pi_i + kT + x_{ij}]$ for $k \in \mathbb{Z}$ and $a = (i, j) \in \mathcal{A}$. Now, an *infrastructure assignment* is any map $\nu\colon \mathcal{I} \to E$, and we say it is *valid* if we have that $\nu\left(I_a^{(k)}\right) \in \eta(a)$ for all $I_a^{(k)} \in \mathcal{I}$. Furthermore, for a given infrastructure assignment ν and an infrastructure element $e \in E$ we define

$$\mathcal{I}_e := \left\{ [\pi_i + kT, \pi_i + kT + x_{ij} + h_e) \;\middle|\; \forall I_a^{(k)} \in \mathcal{I}\colon \nu\left(I_a^{(k)}\right) = e \right\}, \tag{5}$$

and we say ν is *h-conflict-free* if the intervals in $\mathcal{I}_e$ are pairwise disjoint for every $e \in E$.

It now comes natural to formulate the following.

▶ **Definition 4.** *Let (G, T, ℓ, u, w) be a* PESP *instance and $\eta\colon \mathcal{A} \to \mathcal{E} \subseteq 2^E$ an infrastructure map, and let $h \in \mathbb{R}^E_{\geq 0}$. The* Infrastructure-Aware PESP with Assignment (IPESPA) *is to find a solution (π, x) to* PESP *on (G, T, ℓ, u, w), together with a valid and h-conflict-free infrastructure assignment ν, such that the solution is optimal, or to decide that no such solution exists.*

It is clear that were η not to be actually flexible, meaning $|\eta(a)| = 1$ for every $a \in \mathcal{A}$, then we would fall back into Definition 2 by fixing the only possible assignment $\nu(I_a^{(k)}) := \eta(a)$ for all intervals in the realisation. Otherwise, this problem formulation allows for full flexibility in the choice of infrastructure, which can change after any periodic repetition, within the limits of η. This lack of structure and predictability may seem to be an issue of design, since the solutions could even become indescribable in finite terms. Thankfully, this will turn out not to be an issue. We say an infrastructure assignment ν is ω-periodic, for some $\omega \in \mathbb{N}$, if

$$\nu(I_a^{(k)}) = \nu(I_a^{(k)} + \omega T), \tag{6}$$

for every $I_a^{(k)} \in \mathcal{I}$. In such a case, we call ω the infrastructural period of the assignment. As it turns out, we are always able to restrict to such repeating patterns without losing any underlying PESP solution.

▶ **Theorem 5.** *Consider an instance (G, T, ℓ, u, w) of* IPESPA *with $\eta\colon \mathcal{A} \to \mathcal{E} \subseteq 2^E$, and a solution (π, x) together with a valid and h-conflict-free infrastructure assignment ν. Then, there exist $\omega \in \mathbb{N}$, with $\omega \leq |E|^{|E|}$, and a valid and h-conflict-free infrastructure assignment σ such that σ is ω-periodic.*

Proof. Let us consider $\mathcal{I}$ for the above instance and solution. All activities have a single representative interval in $\mathcal{I}$ whose lower bound is in $[pT, pT + T)$, with $p \in \mathbb{Z}$. This set of representatives, which we denote by $\mathcal{F}_p$, is finite. Even more so, considering for each $e \in E$ the one interval in $\mathcal{F}_p$ that is ν-assigned to e and with the minimum lower bound, there are at most $|E|$ such leading intervals. There are at most $|E|^{|E|}$ ways to assign these intervals, and so there exists a $p' \in \mathbb{Z}$ such that $p - |E|^{|E|} \leq p' \leq p + |E|^{|E|}$, and such that $\nu|_{\mathcal{F}_{p'}}$ mirrors $\nu|_{\mathcal{F}_p}$ on all the leading intervals of $\mathcal{F}_p$. By that, mirroring the whole of $\nu|_{\mathcal{F}_p}$ on $\mathcal{F}_{p'}$ can be done without h-conflict. Then, without loss of generality, we assume that $p < p'$, and set $\omega := p' - p$. By construction $\omega \leq |E|^{|E|}$, and we can construct a valid, h-conflict-free, and ω-periodic infrastructure assignment σ by setting

$$\sigma\left(I_a^{(k)}\right) := \nu\left(I_a^{([k-p]_\omega)}\right). \tag{7}$$

◀

The theorem ensures that IPESPA can be solved by restricting to periodic assignments, whose maximum period is bounded by the instance. Note that the bound on ω can be significantly improved if all headways are the same, i.e., $h_e = h_{e'}$ for any $e, e' \in E$. In that case, much along the lines of [6, Theorem 3.1], the bound becomes $\omega \leq |E|!$.

3.2 Pattern Functions and Conflict-Freeness

We will now construct a finitely described object from which a periodic infrastructure assignment can be derived, and conclude this chapter by showing how to use said object to formulate IPESPA as a mixed-integer program. As in Definition 4 we have a PESP instance, an infrastructure map, and a vector of headways. Choosing some maximum infrastructural period $M \in \mathbb{N}_{>0}$, we construct a *pattern function* $\mathcal{H} \colon \mathcal{A} \to \mathcal{P}$, that assigns to each arc in $\mathcal{A}$ a *pattern* in

$$\mathcal{P} := \bigcup_{i|M} E^i = \{(e_1, \ldots, e_i) \mid e_1, \ldots, e_i \in E \text{ and } i \text{ divides } M\}. \tag{8}$$

A pattern function is said to be *valid* if every image $\mathcal{H}(a)$ only contains elements of $\eta(a)$. The prescribed pattern is intended to be repeated ad infinitum. A corresponding infrastructure assignment $\nu_{\mathcal{H}}$ is quickly extracted from a pattern function $\mathcal{H}$, by setting

$$\nu_{\mathcal{H}}\left(I_a^{(k)}\right) := \mathcal{H}(a)_{[k]_{m_a}} \qquad \forall a \in \mathcal{A}, \forall k \in \mathbb{Z}, \tag{9}$$

where m_a is the length of $\mathcal{H}(a)$, i.e., the number of entries. So constructed, $\nu_{\mathcal{H}}$ is periodic, of period at most M. Note that the choice of M is up to the planner and the model becomes more flexible the more M is divisible.

▶ **Example 6.** Let us consider again Example 3 in the case where we allow to assign each activity to each platform, i.e., $\eta(a_k) = \{e_1, e_2\}$ for all $k \in \{1, 2, 3\}$. The valid pattern function associated to the infrastructure assignment in Figure 1b is given by $\mathcal{H}(a_1) = \mathcal{H}(a_3) = (e_1, e_2), \mathcal{H}(a_2) = (e_2, e_1)$, so that $m_{a_1} = m_{a_2} = m_{a_3} = 2$.

We can now use pattern functions to formulate linear modulo constraints, that generalize the Q0-constraints as presented in (2).

▶ **Theorem 7.** *Consider a* PESP *instance* (G, T, ℓ, u, w)*, an infrastructure map* $\eta \colon \mathcal{A} \to \mathcal{E} \subseteq 2^E$*, headways* $h \in \mathbb{R}^E_{\geq 0}$*, and some* PESP *solution* (π, x)*. Let* $\mathcal{H}$ *be a pattern function, and denote by* m_a *the length of the pattern* $\mathcal{H}(a)$*. Then, the infrastructure assignment* $\nu_{\mathcal{H}}$ *is* h*-conflict-free if and only if:*

(a) *For every arc* $a \in \mathcal{A}$ *and infrastructure element* $e \in \mathcal{H}(a)$*, we have*

$$x_a + h_e \leq m_a T. \tag{10}$$

(b) *For every $a_1, a_2 \in \mathcal{A}$, with $a_1 = (i_1, j_1)$ and $a_2 = (i_2, j_2)$, with images under $\mathcal{H}$ both containing the same infrastructure element e at indices p_1 and p_2 respectively, such that either $a_1 \neq a_2$ or $p_1 \neq p_2$, we have*

$$\begin{gathered}[\pi_{i_2} + (p_2 + k_2 m_{a_2})\,T - \pi_{i_1} - (p_1 + k_1 m_{a_1})\,T]_{mT} \geq x_{a_1} + h_e, \\ \forall k_1 \in \left\{0, \dots, \frac{m}{m_{a_1}} - 1\right\}, k_2 \in \left\{0, \dots, \frac{m}{m_{a_2}} - 1\right\},\end{gathered} \tag{11}$$

where $m \coloneqq \operatorname{lcm}(m_{a_1}, m_{a_2})$, and the indexing of the patterns starts at 0.

Proof. ($\Longrightarrow$): If (a) is violated, then $\nu_{\mathcal{H}}\left(I_a^{(p)}\right) = e = \nu_{\mathcal{H}}\left(I_a^{(p+m_a)}\right)$, for p the index of e in $\mathcal{H}(a)$. Then we find in $\mathcal{I}_e$ the intervals

$$\begin{aligned}&[\pi_i + pT, \pi_i + pT + x_{ij} + h_e) \text{ and} \\ &[\pi_i + (p + m_a)T, \pi_i + (p + m_a)T + x_{ij} + h_e),\end{aligned} \tag{12}$$

which intersect, since $\pi_i + pT + x_{ij} + h_e > \pi_i + pT + m_a T$.

Suppose instead (a) holds, and (b) is violated, for some $k_1 \in \{0, \dots, {}^{m}/_{m_{a_1}} - 1\}$ and $k_2 \in \{0, \dots, {}^{m}/_{m_{a_2}} - 1\}$. By construction in (9), note that the images under $\nu_{\mathcal{H}}$ of $I_{a_1}^{(p_1 + k_1 m_{a_1})}$ and $I_{a_2}^{(p_2 + k_2 m_{a_2})}$ are both e, in fact for any integral k_1 and k_2. Since $\pi_v \in [0, T)$ for every $v \in V(G)$, we have that

$$[\pi_{i_2} + (p_2 + k_2 m_{a_2})\,T - \pi_{i_1} - (p_1 + k_1 m_{a_1})\,T]_{mT} < x_{a_1} + h_e, \tag{13}$$

and by construction the content of the modulo operator is either already in $[0, mT)$, or it is in $[-mT, 0)$. If it is non-negative, we find in $\mathcal{I}_e$ the intervals

$$\begin{aligned}&[\pi_{i_1} + (p_1 + k_1 m_{a_1})T, \pi_{i_1} + (p_1 + k_1 m_{a_1})T + x_{a_1} + h_e) \text{ and} \\ &[\pi_{i_2} + (p_2 + k_2 m_{a_2})T, \pi_{i_2} + (p_2 + k_2 m_{a_2})T + x_{a_2} + h_e),\end{aligned} \tag{14}$$

and they intersect. If instead the content is negative, then the modulo operator will add mT, and we find in $\mathcal{I}_e$ the intervals

$$\begin{aligned}&[\pi_{i_1} + (p_1 + k_1 m_{a_1})T, \pi_{i_1} + (p_1 + k_1 m_{a_1})T + x_{a_1} + h_e) \text{ and} \\ &[\pi_{i_2} + (p_2 + k_2 m_{a_2} + m)T, \pi_{i_2} + (p_2 + k_2 m_{a_2} + m)T + x_{a_2} + h_e),\end{aligned} \tag{15}$$

and they intersect.

($\Longleftarrow$): Suppose now that $\nu_{\mathcal{H}}$ is not h-conflict-free. There is then an element $e \in E$ such that the set $\mathcal{I}_e$ contains two intersecting intervals. Let these be $I_{a_1}^{(s_1)}$ and $I_{a_2}^{(s_2)}$, and without loss of generality let us assume that $\min I_{a_2}^{(s_2)} \in I_{a_1}^{(s_1)}$. We then have that

$$0 \leq \pi_{i_2} + s_2 T - \pi_{i_1} - s_1 T < x_{a_1} - h_e \leq m_{a_1} T, \tag{16}$$

where the last inequality holds if (a) does. Then, since $m_{a_1} T \leq mT$, the $[\cdot]_{mT}$ operator is freely applied, and we have $[\pi_{i_2} + s_2 T - \pi_{i_1} - s_1 T]_{mT} < x_{a_1} + h_e$. For both intervals, another way to write s_i is as $p_i + k_i m_{a_i} + \beta_i m$, with integral β_i and minimal positive integral k_i, by which we have $[\pi_{i_2} + (p_2 + k_2 m_{a_2} + \beta_2 m)T - \pi_{i_1} - (p_1 + k_1 m_{a_1} + \beta_1 m)T]_{mT} < x_{a_1} + h_e$. Then, the two summands $\beta_2 mT$ and $\beta_1 mT$ can be deleted since they do not affect the modulo operation, and we have found a violation of (b). ◀

▶ Remark 8. In the IPESP case, i.e., $|\eta(a)| = 1$ for all $a \in \mathcal{A}$, all patterns $\mathcal{H}_a$ have length $m_a = 1$, so that $m = 1$. Theorem 7 then states that an infrastructure assignment is h-conflict-free if and only if $x_a + h_e \leq T$ for all $e \in E$ and $a \in \mathcal{A}_e$ and the Q0-constraint (2) holds for all pairs of distinct arcs assigned to the same infrastructure element. Indeed, this was our definition of h-conflict-freeness in Section 2. In general, the constraints (11) can be interpreted as Q0-constraints that implicitly capture a time expansion up to period mT.

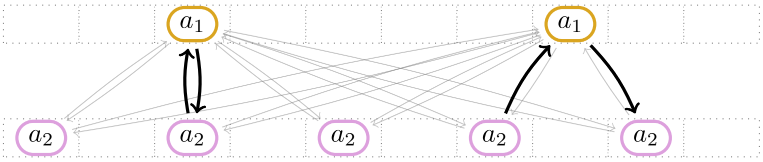


Figure 2 Two time expansions over 10 periods, with occurrences of two activities, a_1 and a_2, with different pattern lengths. Each arc symbolizes one constraint like (19). Only those highlighted in black are needed in case there were upper bounds $u_{a_1}, u_{a_2} \leq T$.

3.3 A MIP Formulation for IPESPA

We now want to use the platforming period bound of Theorem 5 and the inequalities of Theorem 7 to extend the PESP mixed-integer program (1) to IPESPA. We model a pattern function $\mathcal{H}$ by introducing binary variables $\tau_{a\rho}$ for all possible $a \in \mathcal{A}$ and images $\rho \in \mathcal{P} = \bigcup_{i|M} E^i$, whenever ρ is valid for a, i.e., all entries of ρ are in $\eta(a)$. By the bound expressed in Theorem 5 we can choose a finite but sufficiently large M, thereby having finitely many variables $\tau_{a\rho}$. Exactly one pattern has to be activated for each arc, which we express by

$$\sum_{\rho \in \mathcal{P}} \tau_{a\rho} = 1, \qquad \forall a \in \mathcal{A}. \tag{17}$$

We include the inequalities (10), but only activate them if relevant, by having

$$x_a + \max_{e \in \rho} h_e \leq m_a T + (1 - \tau_{a\rho}) B, \tag{18}$$

for every variable $\tau_{a\rho}$. Here $\max_{e\in\rho} h_e$ and m_a are scalars, determined by the pattern ρ, and $B := \max_{a\in\mathcal{A}} u_a + \max_{e\in E} h_e$ is a scalar globally determined by the instance itself. This way, if $\mathcal{H}(a) = \rho$, then (18) is effective, but otherwise it is trivially satisfied.

To linearize the modulo operation in (11) we also introduce binary variables $s_{\rho_1\rho_2}$. These are analogous to the periodic offsets p_{ij} (1e) and can indeed be restricted to $\{0, 1\}$, as in this case the modulo operator is applied to a number in $[-mT, mT)$. Activating the constraint if and only if both patterns ρ_1 and ρ_2 are selected, we have

$$\pi_{i_2} + (p_2 + k_2 m_{a_2}) T - \pi_{i_1} - (p_1 + k_1 m_{a_1}) T + mT s_{\rho_1\rho_2} \geq x_{a_1} + h_e - (2 - \tau_{a_1\rho_1} - \tau_{a_2\rho_2}) B, \tag{19}$$

for every pair of arcs $a_1, a_2 \in \mathcal{A}$, respectively with $\mathcal{H}$-images ρ_1, ρ_2, of lengths m_{a_1}, m_{a_2}, both containing $e \in E$ at indices p_1, p_2 (0-indexed), for every $k_1 \in \{0, \ldots, {}^{m}/_{m_{a_1}} - 1\}$ and $k_2 \in \{0, \ldots, {}^{m}/_{m_{a_2}} - 1\}$, and where $m = \operatorname{lcm}(m_{a_1}, m_{a_2})$. Note that p_i, m_{a_i}, m, and h_e, here are all scalars, determined by ρ_1 and ρ_2.

▶ **Example 9.** For an illustration of which constraints (19) are applied, we refer to Figure 2. There we have two activities a_1 and a_2, assignable to the same infrastructure element, with $p_1 = 2$, $m_1 = 5$, and $p_2 = 0$, $m_2 = 2$. Each arc symbolizes one constraint like (19), of which there is two per pairing, i.e., 20 in total. However, in many cases, if $p_2 + k_2 m_{a_2} - p_1 - k_1 m_{a_1} > \lceil {}^{(u_{a_1} + h_e)}/_{T} \rceil$, then (11) is always satisfied, and we can exclude it a priori. This means that, depending on the PESP upper bounds, a significant drop in the number of constraints is possible. The bold arcs in Figure 2 are the 4 constraints that would be kept if we had, for instance, $u_{a_1}, u_{a_2} \leq T$.

Including in a PESP mixed-integer program such as (1) these two types of binary variables $\tau_{a\rho}$ and $s_{\rho_1\rho_2}$, together with (17), (18), and (19), yields a mixed-integer program formulation for IPESPA. In (20) we can see it in full.

$$\min \sum_{a\in A(G)} w_a x_a \tag{20a}$$

$$\text{s.t.} \quad (\pi, x) \text{ solves PESP on } G, \tag{20b}$$

$$\sum_{\rho\in\mathcal{P}} \tau_{a\rho} = 1 \qquad \forall a \in \mathcal{A}, \tag{20c}$$

$$x_a + \max_{e\in\rho} h_e \le m_a T + (1 - \tau_{a\rho})B \qquad \begin{array}{l}\forall a \in \mathcal{A} \text{ and}\\ \forall \text{ valid } \rho \in \mathcal{P},\end{array} \tag{20d}$$

$$\begin{pmatrix} \pi_{i_2} + (p_2 + k_2 m_{a_2})T \\ -\pi_{i_1} - (p_1 + k_1 m_{a_1})T \\ +mTs_{\rho_1\rho_2} \end{pmatrix} \ge \begin{pmatrix} x_{a_1} + h_e \\ -(2-\tau_{a_1\rho_1} - \tau_{a_2\rho_2})B \end{pmatrix} \qquad \forall a_1, a_2 \text{ s.t. } (\star), \tag{20e}$$

$$\tau_{a\rho} \in \{0, 1\} \qquad \begin{array}{l}\forall a \in \mathcal{A} \text{ and}\\ \forall \text{ valid } \rho \in \mathcal{P},\end{array} \tag{20f}$$

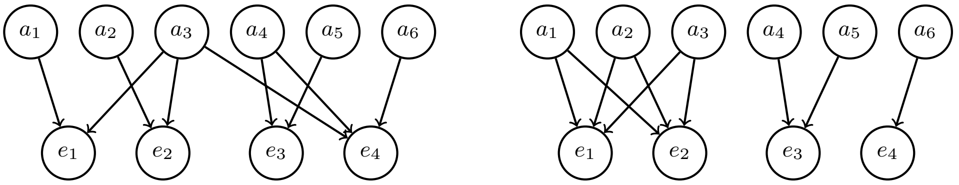
$$s_{\rho_1\rho_2} \in \{0, 1\} \qquad \forall \rho_1, \rho_2 \in \mathcal{P}, \tag{20g}$$

where by $(\star)$ we mean the conditions of (19).

This formulation allows for a great degree of flexibility, giving practitioners a direct handle on the maximum infrastructural period M. In fact, although fixing M to the bound proven in Theorem 5 ensures that no PESP solution is excluded, it is entirely possible that in a practical setting one would want to limit it further, so as to bound the complexity of the infrastructural assignment. To that same purpose, the formulation also allows to forcibly forbid individual patterns if desired.

4 Partitionable Infrastructure Maps

In practice, the infrastructure map $\eta \colon \mathcal{A} \to \mathcal{E}$ is oftentimes not completely flexible, as shown in Figure 3a, but instead comes with additional structure. Of particular interest is the case illustrated in Figure 3b, where $\mathcal{E}$ is a partition of all infrastructure elements, i.e., the infrastructure elements are all grouped and every activity can be freely assigned to all elements in one of such groups. An omnipresent example is a station with two platforms that serves lines in two directions, with the lines in each direction having a dedicated platform. It turns out that when η is *partitionable*, the IPESPA boils down to a rather compact form.



(a) Non-partitionable η.

(b) Partitionable η.

Figure 3 Two infrastructure maps for an instance with six activities and four infrastructure elements.

Formally, we define the following variant of the IPESPA.

▶ **Definition 10.** *Let (G, T, ℓ, u, w) be a* PESP *instance and $\eta\colon \mathcal{A} \to \mathcal{E}$ an infrastructure map, where $\mathcal{E}$ is a partition of E. The* Infrastructure-Aware PESP with Capacities (IPESPC) *is to find a solution (π, x) to* PESP *on (G, T, ℓ, u, w), together with a valid and conflict-free platform assignment ν, such that the solution is optimal, or to decide that no such solution exists.*

Alternatively, IPESPC is equivalent to IPESP with the additional feature that every $e \in E$ has now a capacity $k_e \in \mathbb{N}$. In other words, $e \in E$ no longer corresponds to a single infrastructure element, but to a group of k_e equivalent elements. For ease of exposition, we stick to this perspective going forward.

We use a matching-based approach to solve IPESPC. To this end, we expand G with auxiliary arcs between activities that can use the same group of infrastructure elements. Formally, let G' be the graph arising from G by adding for each $e \in E$ and $a_1 = (i_1, j_1), a_2 = (i_2, j_2) \in \mathcal{A}_e$ a new arc α from j_1 to i_2. We refer to α as the auxiliary arc from a_1 to a_2, and much like the headway arcs a^{I} used in [2] and in the Q4 butterfly constraints (3), we set $\ell_\alpha := h_e$, $u_\alpha := T - h_e$, $w_\alpha := 0$ if $a_1 \neq a_2$, and $\ell_\alpha := h_e, u_\alpha := T, w_\alpha := 0$ in the case $a_1 = a_2$. Let $\mathcal{A}'$ denote the set of all auxiliary arcs, let $\mathcal{A}'_e$ denote all auxiliary arcs associated to e, and let G'_e be the subgraph of G' on the arcs in $\mathcal{A}'_e$. For $S \subseteq V(G) \times V(G)$, we will use the notation $G[S]$ for the graph $(V(G), A(G) \cup S)$, so that, e.g., $G' = G[\mathcal{A}']$. The following theorem compactly characterizes when a PESP solution admits a feasible infrastructure assignment in the context of IPESPC:

▶ **Theorem 11.** *Consider a* PESP *instance (G, T, ℓ, u, w), the expanded graph G', a partitionable infrastructure map $\eta\colon \mathcal{A} \to \mathcal{E}$, headways $h \in \mathbb{R}^E_{\geq 0}$, capacities $k \in \mathbb{N}^E$, and some* PESP *solution (π, x) on G. Then, there exists a valid and h-conflict-free infrastructure assignment ν if and only if for each $e \in E$ there exists a perfect matching $\mathcal{M}'_e \subseteq \mathcal{A}'_e$ of G'_e and (π, x) can be extended to a* PESP *solution on $G[\mathcal{M}'_e]$ such that*

$$\sum_{a \in \mathcal{A}_e} x_a + \sum_{a \in \mathcal{M}'_e} x_a \leq k_e T. \tag{21}$$

Proof. First, assume that there exist perfect matchings $\mathcal{M}'_e$ of G'_e satisfying (21) for all $e \in E$. The matching $\mathcal{M}'_e$ together with the arcs $\mathcal{A}_e$ forms a set of disjoint directed cycles, where every cycle consists of arcs that alternatingly belong to $\mathcal{A}_e$ and $\mathcal{A}'_e$.

Let then $C^e_1, C^e_2, \dots, C^e_{m_e}$ denote the cycles corresponding to $e \in E$, and define $p^e_j := \frac{1}{T} \sum_{a \in C^e_j} x_a$ for $j = 1, \dots, m_e$. Because the timetable (π, x) is feasible on $G[\mathcal{M}'_e]$, p^e_j is integer by the cycle periodicity property [7, Lemma 6.39]. Since it holds that

$$\sum_{j=1}^{m_e} p^e_j = \frac{1}{T} \sum_{j=1}^{m_e} \sum_{a \in C^e_j} x_a = \frac{1}{T} \left(\sum_{a \in \mathcal{A}_e} x_a + \sum_{a \in \mathcal{M}'_e} x_a \right) \leq k_e, \tag{22}$$

it suffices to show that just the activities appearing in C^e_j can be assigned on a group of capacity p^e_j. The total tension along the cycle is $p^e_j T$, so in a $p^e_j T$-periodic schedule it is straightforward to fit all the activities in $C^e_j \cap \mathcal{A}$, simply following the order in which they appear through the cycle and with timestamps agreeing with π modulo T. Then, having an available capacity of p^e_j, that schedule can be repeated over each of the p^e_j equivalent elements, each time shifted forward by T. This construction implies adherence to the T-periodic timetable π, and a $p^e_j T$-periodic infrastructure assignment, valid since each $\eta(C^e_j \cap \mathcal{A}) = e$, and h-conflict-free by the bounds on the auxiliary arcs.

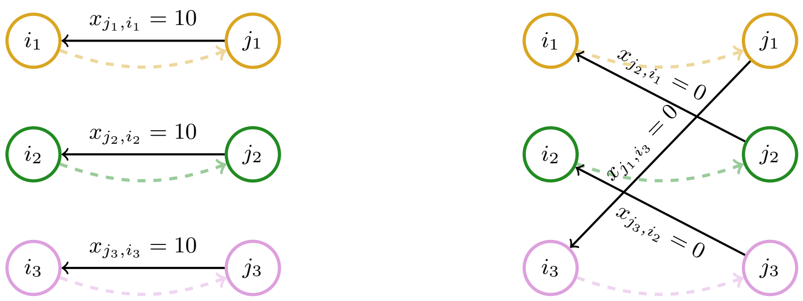
Suppose the contrary instead, that no perfect matching in $\mathcal{A}'_e$ respects (21). Notice how, given that π is fixed and the headway h_e is the same on all elements in the group e, we can without loss of generality lengthen all activities in $\mathcal{A}_e$ by h_e and then assume that the new minimum headway is 0 instead. Now, let $\mathcal{M}^*$ be a minimum tension perfect matching in G'_e. We then have that for some $e \in E$ there is an integer $K_e > k_e$ such that

$$\sum_{a \in \mathcal{A}_e} x_a + \sum_{a \in \mathcal{M}^*} x_a = K_e T > k_e T. \tag{23}$$

Now, for $t \in [0, T)$ and $S \subseteq \mathcal{A}_e \cup \mathcal{A}'_e$, let the *inventory function* $I_e(t, S)$ denote the number of h-overlapping activities in S at time t. By (23) we have that $I_e(t, \mathcal{A}_e \cup \mathcal{M}^*) = K_e$ for all $t \in [0, T)$. Moreover, since $\mathcal{M}^*$ is a minimum tension perfect matching, then there is a $t^* \in [0, T)$ such that $I_e(t^*, \mathcal{M}^*) = 0$. That is because if $\mathcal{M}^* = \{(e_1, f_1), \ldots, (e_m, f_m)\}$ was h-overlapping everywhere instead, then without loss of generality we can assume that the timestamps would be

$$\pi_{e_1} < \pi_{f_m} < \pi_{e_2} < \pi_{f_1} < \ldots < \pi_{e_m} < \pi_{f_{m-1}}, \tag{24}$$

where a shorter matching is immediately apparent, negating the minimality of $\mathcal{M}^*$. See [17, Lemma 3] for further details. By the above, we then have that $I_e(t^*, \mathcal{A}_e) = I_e(t^*, \mathcal{A}_e \cup \mathcal{M}^*) - I_e(t^*, \mathcal{M}^*) = K_e > k_e$, implying there are more simultaneous activities than there is infrastructure capacity to host them. In particular, there is no h-conflict-free infrastructure assignment. ◀



(a) Matching corresponding to Figure 1a. **(b)** Matching corresponding to Figure 1b.

Figure 4 The two matchings corresponding to Example 3.

▶ **Example 12.** We illustrate on the basis of Example 3 how matchings relate to infrastructure assignments in Figure 4. In the IPESPA situation of Figure 1a, we have three infrastructure elements e_1, e_2, e_3 of capacity one each. For each element e_k, the matching in Figure 4a creates a directed cycle of tension $1 \cdot T = 30$ containing the dwelling activity $a_k = (i_k, j_k)$ and the auxiliary arc (j_k, i_k). Figure 1b is an IPESPC situation with one infrastructure element e. Here, the matching in Figure 4b induces a directed cycle of tension $2 \cdot T = 60$, we hence use a capacity of two. The cycle encodes that each platform is used by a_1, a_3, a_2 in this cyclic order.

Theorem 11 allows formulating IPESPC very compactly compared to IPESPA. Introducing matching variables as binary decision variables y_a for all $a \in \mathcal{A}'$, we have the following MIP:

$$\min \sum_{a \in A(G)} w_a x_a \tag{25a}$$

$$\text{s.t.} \quad (\pi, x) \text{ solves PESP on } G, \tag{25b}$$

$$\sum_{a \in \delta^-_{G'_e}(i)} y_a = 1 \qquad \forall e \in E, \forall (i,j) \in \mathcal{A}_e, \tag{25c}$$

$$\sum_{a \in \delta^+_{G'_e}(j)} y_a = 1 \qquad \forall e \in E, \forall (i,j) \in \mathcal{A}_e, \tag{25d}$$

$$\sum_{a \in \mathcal{A}_e} x_a + \sum_{a \in \mathcal{A}'_e} x_a y_a \leq k_e T \qquad \forall e \in E, \tag{25e}$$

$$\pi_j - \pi_i + T p_a = x_a \qquad \forall a = (i,j) \in \mathcal{A}', \tag{25f}$$

$$\ell_a y_a \leq x_a \leq u_a y_a + (T-1)(1-y_a) \quad \forall a \in \mathcal{A}', \tag{25g}$$

$$y_a \in \{0,1\} \qquad \forall a \in \mathcal{A}'. \tag{25h}$$

Constraints (25c) and (25d) define a unique predecessor and successor for each activity by requiring that y corresponds to a perfect matching $\mathcal{M}'_e$ in $\mathcal{A}'_e$ for each $e \in E$. Constraints (25e) ensure that the total time of the activities scheduled on a platform group and the selected auxiliary activities is at most the total available time on that platform group. These constraints can be linearized by introducing for each $a \in \mathcal{A}'$ a real variable z_a, with bounds $0 \leq z_a \leq u_a$, and constrained as $x_a - (1-y_a)u_a \leq z_a \leq x_a$, so that it is equal to the product $x_a y_a$. The constraints (25f) tie the tensions x_a on the auxiliary activities $a \in \mathcal{A}'$ to the timetable π. Finally, (25g) ensures that those tensions adhere to their bounds when they are part of the selected matching, and impose no restrictions otherwise.

There is a clear resemblance between (25) and the formulation [17] proposes for jointly optimizing a periodic timetable and vehicle circulation. This is no coincidence: as long as the corresponding mapping from activities to resources is a partition, any resource schedule associated to a periodic timetable can be described by a matching. In [17] the resources are vehicles, whereas in the present paper infrastructure elements, e.g., platforms. It immediately follows that the results established in [17] carry over to our setting. Most notably, given a feasible timetable, a greedy algorithm can actually be used to find an infrastructure assignment with the minimum number of required infrastructure elements.

For groups consisting of a single element, i.e., $k_e = 1$, the matching formulation can be enhanced using the surprising insight that in this case it is not necessary to compute the matching explicitly. We have the following theorem:

▶ **Theorem 13.** *Suppose $k_e = 1$ and let (π, x) be a* PESP *solution on G. Then $\mathcal{A}_e$ is h-conflict-free if and only if (π, x) extends to a* PESP *solution on G'_e such that*

$$\sum_{a \in \mathcal{A}_e} x_a + \left(\frac{1}{|\mathcal{A}_e|} \sum_{a \in \mathcal{A}'_e} x_a \right) = \frac{|\mathcal{A}_e| + 1}{2} T. \tag{26}$$

Proof. Suppose that $\mathcal{A}_e$ is h-conflict-free. This means that for any $a_s = (i_s, j_s), a_t = (i_t, j_t) \in \mathcal{A}_e$ with $a_s \neq a_t$ the Q4-constraints (3) must hold, namely

$$x_{i_s,j_s} + x'_{j_s,i_t} + x_{i_t,j_t} + x'_{j_t,i_s} = T, \tag{27}$$

where we use the notation $x'_a := [\pi_j - \pi_i - \ell_a]_t + \ell_a$ to indicate tensions on an auxiliary arc $a = (i,j)$.

Let $q(a_s, a_t)$ denote the butterfly-shaped 4-cycle given by $(i_s, j_s, i_t, j_t, i_s)$. There are $\sum_{i=1}^{|\mathcal{A}_e|-1} i = |\mathcal{A}_e|(|\mathcal{A}_e|-1)/2$ many such butterfly cycles, each auxiliary arc $(j_s, i_t) \in \mathcal{A}'_e$ with $s \neq t$ is in exactly one such cycle, while each arc $(i_s, j_s) \in \mathcal{A}_e$ is in $|\mathcal{A}_e| - 1$ many of them. Moreover, there are exactly $|\mathcal{A}_e|$ auxiliary arcs of the form (j_s, i_s), for which holds that

$$h_e \leq x_{i_s,j_s} + x'_{j_s,i_s} \leq 2T - h_e, \tag{28}$$

since $h_e \leq x'_{j_s,i_s} \leq T$ and h-conflict freeness implies $x_{i_s,j_s} \leq T - h_e$. Due to the cycle periodicity property,

$$x_{i_s,j_s} + x'_{j_s,i_s} = T, \tag{29}$$

unless $h_e = 0$, but then we can subtract T from $x'_{j_s,i_s} = T$ and maintain the feasibility of (π, x) on G'_e.

Summing up over all butterfly constraints (27) for each pair of activities and all the self-cycles (29) of each activity in $\mathcal{A}_e$, we obtain

$$|\mathcal{A}_e| \sum_{s=1}^{|\mathcal{A}_e|} x_{i_s,j_s} + \sum_{a \in \mathcal{A}'_e} x'_a = \left(\frac{|\mathcal{A}_e|(|\mathcal{A}_e|-1)}{2} + |\mathcal{A}_e| \right) T = \frac{|\mathcal{A}_e|(|\mathcal{A}_e|+1)}{2} T. \tag{30}$$

For the other direction, suppose that $\mathcal{A}'_e$ is not h-conflict free, but (π, x) extends to PESP solution on G'_e. Then one of (27) or (29) must be violated. Let $q(a_s, a_t)$ be a butterfly cycle with $a_s = (i_s, j_s)$ and $a_t = (i_t, j_t)$. Then

$$x_{i_s,j_s} + x'_{j_s,i_t} + x_{i_t,j_t} + x'_{j_t,i_s} \geq \ell_{i_s,j_s} + \ell_{i_t,j_t} + 2h_e > 0 \tag{31}$$

since we assumed that at least lower bounds or minimum headway times are positive. Due to the cycle periodicity property of periodic timetables,

$$x_{i_s,j_s} + x'_{j_s,i_t} + x_{i_t,j_t} + x'_{j_t,i_s} \geq T. \tag{32}$$

Moreover, considering the cycles comprised of $(i_s, j_s) \in \mathcal{A}_e$ and the auxiliary arc $(j_s, i_s) \in \mathcal{A}'_e$, we have

$$x_{i_s,j_s} + x'_{j_s,i_s} \geq \ell_{i_s,j_s} + h_e > 0, \tag{33}$$

so that

$$x_{i_s,j_s} + x'_{j_s,i_s} \geq T. \tag{34}$$

Therefore, if one of (27) or (29) is violated, we must have a strict inequality in (32) or (34). Taking the sum,

$$|\mathcal{A}_e| \sum_{s=1}^{|\mathcal{A}_e|} x_{i_s,j_s} + \sum_{a \in \mathcal{A}'_e} x'_a > \left(\frac{|\mathcal{A}_e|(|\mathcal{A}_e|-1)}{2} + |\mathcal{A}_e| \right) T = \frac{|\mathcal{A}_e|(|\mathcal{A}_e|+1)}{2} T, \tag{35}$$

so (26) cannot hold. ◀

A direct implication of Theorem 13 is a new formulation for IPESP, provided that all infrastructure elements have unit capacities:

$$\min \quad \sum_{a \in A(G)} w_a x_a \tag{36a}$$

$$\text{s.t.} \quad (\pi, x) \text{ solves PESP} \quad \text{on } G', \tag{36b}$$

$$\sum_{a \in \mathcal{A}_e} x_a + \frac{1}{|\mathcal{A}_e|} \sum_{a \in \mathcal{A}'_e} x_a = \frac{|\mathcal{A}_e|+1}{2} T \quad \forall e \in E. \tag{36c}$$

In contrast to the original formulation proposed in [2] and to the matching approach (25), formulation (36) introduces neither additional integer variables nor quadratic terms.

5 Experiments

In the sequel, we will evaluate different formulations for IPESP and IPESPC on a set of realistic instances. We omit the IPESPA model, as for our datasets there hardly is added value compared to IPESPC. We use Gurobi 11 as a MIP solver on an Intel Xeon E3-1270 3.80 GHz CPU with 32 GB RAM.

5.1 Instances

We evaluate our models on instances we constructed based on publicly available timetable information, platform usage, and track data. Additionally, some track information was provided to us by DB InfraGO AG. The instances are:

- `S-Bahn`, the full network of S-Bahn Berlin, a suburban commuter rail network with 16 lines. On several sections, there are as much as 7 trains per track and direction within the period time of 20 minutes. Our IPESP instance is based on the annual timetable, assuming fixed driving times, but flexible dwelling and turnaround times. However, there are several places in the network where multiple platforms are available, and this builds our corresponding IPESPC instance.
- `Tram`, the full tram network of Berlin, comprising 22 lines operated with a period time of 20 minutes, with frequencies ranging between 1 and 6. The difficulty here does not lie in associating driving and dwelling times, which are fixed, but in fulfilling synchronization constraints and deciding infrastructure assignments at the terminal stations: The turnaround times are flexible and capacities in the turning loops are scarce. This is inherently an IPESPC instance, that, in fact, cannot be transformed into a feasible IPESP instance, since all T-periodic assignments are infeasible.
- `Corridor`, the central longitudinal railway corridor of regional and long-distance trains in Berlin, as well as a subsection of it, from Ostkreuz to Friedrichstraße, which we denote as `ShortCorridor`. Many stations have multiple platforms, and the trains have different stopping patterns. The period time is 60 minutes, driving, dwelling, and turnaround times are all variable. This, too, is inherently an IPESPC instance, from which we created an IPESP instance based on the annual timetable.

We use only a simple objective function for the timetabling part: Driving and dwelling activities are weighted by 2, turnarounds by 1, and all other arcs by 0. Additionally, note that we do not include any transfer arcs, in part because we have no data available regarding the flow of passengers, and in part because the scope of this work rather focuses on operational capabilities instead.

5.2 IPESP Experiments

For the unit capacity case of IPESP, we have now several formulations at hand: The Q4 butterfly constraints (3), the matching model (25), and the special formulation (36). We test these formulations and their combinations on the `S-Bahn` and `Corridor` instance, with a wall time limit of one hour. We further include versions where the matching variables y are relaxed to be continuous. Our results are collected in Table 1.

On `S-Bahn`, not all formulations found a solution, but those that did also managed to prove optimality within the time limit. All such formulations contain the Q4 butterfly constraints (3), and none of them use binary matching variables. The combination of Q4 with the special sum constraint (36) worked best, followed by pure Q4. When initialized with the optimal solution as MIP start, almost all formulations proved its optimality within the time frame of one hour, or managed a very thin optimality gap.

On Corridor, no formulation found solutions, and no attempt at providing initial partial solutions was successful. On ShortCorridor all formulations quickly found a primal solution within seconds, and an optimal solution within minutes, but none managed to prove that optimality within one hour. Only when starting ShortCorridor with the best solution found so far, a proof to optimality was reached, and only by the formulation using (25), i.e., matching with binary variables, together with the special constraints (36c). This was also the fastest formulation to reach optimality to begin with.

Table 1 Timed results for IPESP tests on S-Bahn and Corridor, expressed in seconds. Tests denoted with Q4 use Q4-constraints as in (3). Tests denoted with M use matching constraints as in (25), with capacity set to 1. Tests denoted with Mc use the same constraints as M, but with relaxed continuous variables instead. Tests denoted with S use constraints as in (36). The first column details the time to the first primal solution that was found, the second column the time to the optimal objective value (3058 for S-Bahn and 10 for Corridor), and the third column the time to fully close the optimality gap. The last column details the time needed to prove optimality when given the optimal solution from the beginning. The time limit was 1 hour per configuration.

S-Bahn	s to primal	s to optimal	s to proof	s to proof (warm)
Q4	64	227	227	95
Q4+M	–	–	–	1221
Q4+Mc	338	1672	1672	140
Q4+S	40	156	156	150
Q4+M+S	–	–	–	– (.58%)
Q4+Mc+S	172	738	738	766
M	–	–	–	– (.61%)
S	–	–	–	595
M+S	–	–	–	2952
Mc+S	–	–	–	1402

ShortCorridor	s to primal	s to optimal	s to proof	s to proof (warm)
Q4	0	110	–	–
Q4+M	5	87	–	–
Q4+Mc	0	82	–	–
Q4+S	2	48	–	–
Q4+M+S	1	15	–	–
Q4+Mc+S	0	23	–	–
M	0	44	–	–
S	1	576	–	–
M+S	4	11	–	3268
Mc+S	6	50	–	–

5.3 IPESPC Experiments

For the instances with capacities larger than one we used formulation (25). To aid the solution process, all infrastructure that still had capacity one has been modelled using Q4 constraints, and used the matching variables where necessary, i.e., for all larger infrastructure. We tested this formulation on the S-Bahn, the Tram, and the Corridor instance, with a time limit of four hours.

On `S-Bahn` no primal solution was found, but we were able to warm start the models with solutions found in the corresponding IPESP tests instead. In that case, the more flexible IPESPC within seconds improved the optimal IPESP solution (albeit by a measly 1.1%), and within approximately 4 more minutes reached the final primal value. In the course of the first hour, Gurobi managed to reduce the optimality gap to 0.2%, where it remained until the time limit.

On `Tram` proven optimality was reached within 10 seconds. Notably, however quick to solve this instance was, it is infeasible to formulate with simple IPESP. Only using the higher capacities enabled by IPESPC it was at all possible to generate a feasible solution.

On `Corridor`, again, no solution was found, but providing a partial starting solution on just the section of `ShortCorridor` was enough to be completed to a full solution for the whole network, which then was improved to proven optimality in only 2 minutes and 25 seconds. On `ShortCorridor` the first primal was found in under 10 minutes and was directly proven to be optimal. Notably, its objective value was better than the IPESP case, now reaching 0 slack. Starting the same test with a solution from the IPESP case reached proven optimality in 3 seconds.

6 Conclusion

This work extends the Infrastructure-Aware PESP (IPESP) framework. One of our new problem formulations, Infrastructure-Aware PESP with Assignment (IPESPA), does so by integrating the choice of the infrastructure assignment within IPESP, allowing for more flexible use of the available infrastructure. In fact, this flexibility can lead to higher efficiency, as well as improved timetables. Although extremely general in its assumptions, we prove that IPESPA can be formulated as a mixed integer linear program (20).

Moving on to a more restricted, but highly realistic scenario, we consider the case when infrastructure elements can be effectively considered as equivalent, and formulate this special version of IPESPA as well, namely Infrastructure-Aware PESP with Capacities (IPESPC). In this case the assignment structure is of note, since it can be seen as a matching problem on a complete bipartite graph, connecting the ends of activities to the starts of the next ones. This gives us not only a compact mixed integer linear program formulation (25), but also novel formulations for IPESP itself, seen as a case of IPESPC with only unit capacities.

Finally, on the practical side, we tested the new matching-based IPESP formulations, as well as the IPESPC formulation. On the unit capacity side, our tests went through various combinations of approaches, and although caution is advised when drawing conclusions, it seems that on the `S-Bahn` instance the Q4-based formulations fared better, whereas on `ShortCorridor`, which has a higher density of larger infrastructure elements to deal with, matching-based formulations had more success. With instances of larger capacity, instead, our tests show that our modelling approach can be of interest in real-world scenarios.

For future work, on the theoretical side we suggest proving tighter bounds on the maximum platforming period, as well as trying to generalize the Q4-constraints much like we here generalized the Q0-constraints. On the practical side, we suggest an iterative approach that uses a separate matching solver to concurrently feed the main model with partial solutions, and to develop heuristic approaches to quickly generate initial solutions.

References

1 E. Bortoletto, N. Lindner, and B. Masing. Tropical Neighbourhood Search: A New Heuristic for Periodic Timetabling. In M. D'Emidio and N. Lindner, editors, *22nd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2022)*, volume 106 of *Open Access Series in Informatics (OASIcs)*, pages 3:1–3:19, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/OASIcs.ATMOS.2022.3.

2 E. Bortoletto, N. Lindner, and B. Masing. Periodic Timetabling with Cyclic Order Constraints. In D. Frigioni and P. Schiewe, editors, *23rd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2023)*, volume 115 of *Open Access Series in Informatics (OASIcs)*, pages 7:1–7:18, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/OASIcs.ATMOS.2023.7.

3 G. Caimi, L. Kroon, and C. Liebchen. Models for railway timetable optimization: Applicability and applications in practice. *Journal of Rail Transport Planning & Management*, 6(4):285–312, 2017. doi:10.1016/j.jrtpm.2016.11.002.

4 D. Frigioni and P. Schiewe, editors. *OASIcs, Volume 115, ATMOS 2023, Complete Volume*, volume 115 of *Open Access Series in Informatics (OASIcs)*, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/OASIcs.ATMOS.2023.

5 F. Fuchs, A. Trivella, and F. Corman. Enhancing the interaction of railway timetabling and line planning with infrastructure awareness. *Transportation Research Part C: Emerging Technologies*, 142:103805, September 2022. doi:10.1016/j.trc.2022.103805.

6 J. Korst, E. Aarts, J. K. Lenstra, and J. Wessels. Periodic multiprocessor scheduling. In E. H. L. Aarts, J. van Leeuwen, and M. Rem, editors, *Parle '91 Parallel Architectures and Languages Europe*, pages 166–178, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg. doi:10.1007/978-3-662-25209-3_12.

7 C. Liebchen. Periodic timetable optimization in public transport. In K.-H. Waldmann and U. M. Stocker, editors, *Operations Research Proceedings 2006*, pages 29–36, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. doi:10.1007/978-3-540-69995-8_5.

8 C. Liebchen and R. H. Möhring. The Modeling Power of the Periodic Event Scheduling Problem: Railway Timetables — and Beyond. In F. Geraets, L. Kroon, A. Schoebel, D. Wagner, and C. D. Zaroliagis, editors, *Algorithmic Methods for Railway Optimization*, Lecture Notes in Computer Science, pages 3–40, Berlin, Heidelberg, 2007. Springer. doi:10.1007/978-3-540-74247-0_1.

9 N. Lindner and B. Masing. On the Split Closure of the Periodic Timetabling Polytope, 2023. doi:10.48550/arXiv.2306.02746.

10 B. Masing, N. Lindner, and C. Liebchen. Integrating Line Planning for Construction Sites into Periodic Timetabling via Track Choice. In D. Frigioni and P. Schiewe, editors, *23rd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2023)*, volume 115 of *Open Access Series in Informatics (OASIcs)*, pages 5:1–5:15, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/OASIcs.ATMOS.2023.5.

11 B. Masing, N. Lindner, and C. Liebchen. Periodic timetabling with integrated track choice for railway construction sites. *Journal of Rail Transport Planning & Management*, 28:100416, 2023. doi:10.1016/j.jrtpm.2023.100416.

12 K. Nachtigall. *Periodic Network Optimization and Fixed Interval Timetables*. Habilitation Thesis, Universität Hildesheim, 1998.

13 K. Nachtigall and J. Opitz. Solving Periodic Timetable Optimisation Problems by Modulo Simplex Calculations. In M. Fischetti and P. Widmayer, editors, *8th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems (ATMOS'08)*, volume 9 of *OpenAccess Series in Informatics (OASIcs)*, Dagstuhl, Germany, 2008. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/OASIcs.ATMOS.2008.1588.

14 A. Hickford S. Blainey and J. Preston. Barriers to passenger rail use: A review of the evidence. *Transport Reviews*, 32(6):675–696, 2012. doi:10.1080/01441647.2012.743489.

15 P. Schiewe and A. Schöbel. Periodic Timetabling with Integrated Routing: Toward Applicable Approaches. *Transportation Science*, 54(6):1714–1731, 2020. Publisher: INFORMS. doi:10.1287/trsc.2019.0965.

16 P. Serafini and W. Ukovich. A mathematical model for periodic scheduling problems. *SIAM J. on Discret. Math.*, 2(4):550–581, 1989. doi:10.1137/0402049.

17 R. N. Van Lieshout. Integrated periodic timetabling and vehicle circulation scheduling. *Transportation Science*, 55(3):768–790, 2021. doi:10.1287/trsc.2020.1024.

Balanced Assignments of Periodic Tasks

Héloïse Gachet
École nationale des ponts et chaussées, Champs-sur-Marne, France
SNCF, DTIPG, Saint-Denis, France

Frédéric Meunier
École nationale des ponts et chaussées, Champs-sur-Marne, France

Abstract

This work deals with a problem of assigning periodic tasks to employees in such a way that each employee performs each task with the same frequency in the long term. The motivation comes from a collaboration with the main French railway company, the SNCF. An almost complete solution is provided under the form of a necessary and sufficient condition that can be checked in polynomial time. A complementary discussion about possible extensions is also proposed.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Applied computing → Transportation

Keywords and phrases Fair schedule, Eulerian digraph, Markov chain, interval graph

Digital Object Identifier 10.4230/OASIcs.ATMOS.2024.5

Funding This research was partially supported by the SNCF as part of a CIFRE PhD.

Acknowledgements The authors are also grateful to Jean-François Delmas for pointing out the situation of "normal numbers," whose existence status shares some similarity with that of balanced feasible assignments; see Section 4.1.

1 Introduction

1.1 Context

The problem studied in this paper was suggested to the authors through a collaboration with the SNCF, the main French railway company. The schedules of their freight train drivers are always periodic: a collection of round trips is repeated every week, with each round trip performed at the same time within the week. Such schedules are often termed "cyclic rosters" in the literature. One motivation for this periodicity is that such schedules are easily understood and memorized by the employees. Another motivation is that these schedules balance experience: in the long term, each round trip is performed the same number of times by each employee. This ensures fairness and also maintains a similar level of proficiency among the employees.

More generally, in the transport sector, periodicity is an important requirement, to which a full body of research is devoted; see, e.g., [5, 7, 8]. Based on the authors' experience and the literature, the concern of balancing experience among employees, given tasks that must be performed periodically, is not only present at the SNCF but also in many other companies. For instance, in an article by Breugem, Dollevoet, and Huisman [2], the same motivations as described in the previous paragraph apply, justifying the use of cyclic rosters among teams of employees (grouped by characteristics) for the Netherlands Railways. Another example, this time for bus drivers, is studied in an article by Xie and Suhl [10].

This raises a natural mathematical question: given tasks that need to be repeated every week and a group of employees, under what conditions is it possible to create (not necessarily periodic) schedules ensuring that in the long term each task is performed the same number of times by each employee?

24th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2024).
Editors: Paul C. Bouman and Spyros C. Kontogiannis; Article No. 5; pp. 5:1–5:12
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this paper, we propose an almost complete solution to this problem. To the authors' knowledge, this problem has not been addressed in the literature. Nevertheless, problems with almost the same input but where the "balancedness" criterion is replaced by a more standard optimization criterion, such as minimizing the number of employees, have been studied in various papers. As an example, Korst, Aarts, Lenstra, and Wessels [4] consider the problem of assigning periodic operations, with fixed starting times and different periods, to a minimal number of processors.

1.2 Problem formulation

Consider a collection of tasks that have to be performed periodically (typically every week in an industrial setting), and a group of indistinguishable employees who will perform them. Formally, we are given

- a collection of n intervals $[a_i, b_i) \subset (-1, 1)$, with $b_i \in (0, 1]$ and $b_i - a_i \leqslant 1$.
- a positive integer q.

Each interval of this collection represents a task: the rth occurrence of task i ($r \in \mathbb{Z}_{>0}$) takes place over the time interval $[a_i + r, b_i + r)$. The number q corresponds to the number of employees, whom we identify from now on with the set $[q]$. Every occurrence of each task has to be assigned to an employee. Such an assignment is *feasible* if no employee is assigned two occurrences overlapping within $\mathbb{R}_{>0}$. Such an assignment is *balanced* if each task is performed by each employee every q periods in the long term average.

In symbols, consider an assignment $f\colon [n] \times \mathbb{Z}_{>0} \to [q]$, where $f(i, r) = j$ means that the rth occurrence of task i is assigned to employee j. It is feasible if

$$[a_i + r, b_i + r) \cap [a_{i'} + r', b_{i'} + r') \neq \varnothing \quad \Longrightarrow \quad f(i, r) \neq f(i', r') \tag{1}$$

for all $i \neq i'$ and all r, r'. (Remark that the left-hand side holds only if $|r - r'| \leqslant 1$.) It is balanced if

$$\lim_{t \to +\infty} \frac{1}{t} \big|\{r \in [t]\colon f(i, r) = j\}\big| = \frac{1}{q}\,, \tag{2}$$

for all $i \in [n]$ and all $j \in [q]$. An illustration is given in Figure 1.

We aim at identifying conditions under which there exists a balanced feasible assignment and at studying the related algorithmic question.

A few comments are in order. First, remark that there exists a feasible assignment if and only if there is no point in $\mathbb{R}$ contained in more than q intervals $[a_i + r, b_i + r)$. (This has also been noted by Korst, Aarts, Lenstra, and Wessels [4, Theorem 2.3], in a more general setting.) This means that for our problem, feasibility is not the challenge. Second, when there is a point of $[0, 1)$ contained in no interval $[a_i + r, b_i + r)$, then the construction of a balanced feasible assignment is trivial: without loss of generality, this point is 0, and any feasible assignment f and any cyclic permutation π of $[q]$ provides a balanced feasible assignment g, periodic with period q, defined by $g(i, r) \coloneqq (\pi^r \circ f)(i, 1)$ for $i \in [n]$ and $r \in \mathbb{Z}_{>0}$. Finally, there are feasible assignments for which the limit in (2) is not well-defined. By definition, if the limit is not well-defined, then the assignment is not balanced.

1.3 Main results

Clearly, a necessary condition for the existence of a balanced feasible assignment is that there is a feasible assignment in which an employee performs each task at least once. Our first main result states the following surprising fact: this condition is actually sufficient.

▶ **Theorem 1.** *There exists a balanced feasible assignment if and only if there exists a feasible assignment with an employee performing each task at least once. Moreover, if there exists a balanced feasible assignment, then there exists such an assignment that is periodic.*

An assignment f is *periodic* if there exists $h \in \mathbb{Z}_{>0}$ such that $f(i, r+h) = f(i, r)$ for all $i \in [n]$ and $r \in \mathbb{Z}_{>0}$. The proof will actually make clear that, in case of the existence of a balanced feasible assignment, it is always possible to get a period h upper-bounded by $q^2 \times q!$.

The necessary and sufficient condition of Theorem 1 is simple enough to obtain an algorithmic counterpart. This is the second main result of the paper.

▶ **Theorem 2.** *Deciding whether there exists a balanced feasible assignment can be done in polynomial time. Moreover, if the number of employees is constant, then such an assignment can be computed in polynomial time when it exists.*

The proofs of these two theorems can be found in Section 3.2. They essentially consist in reducing the question of existence of a balanced feasible assignment to a problem of pebbles on an arc-colored Eulerian directed graph. The latter problem is dealt with in Section 2, which can be read independently of the rest of the paper. Section 3.1 introduces preliminary results and tools, such as a graph $D^{\mathcal{F}}$ built from a well-chosen set $\mathcal{F}$ of feasible assignments and that plays an important role in the proofs. This graph $D^{\mathcal{F}}$ is a particular arc-colored Eulerian directed graph on which we apply the results of Section 2.

2 A problem of pebbles on an arc-colored Eulerian directed graph

This section introduces a problem of pebbles moving on an Eulerian directed graph, which we believe to be interesting for its own sake. The proof of Theorem 1 will essentially consist in reducing the problem of existence of a balanced feasible assignment to this pebble problem. This pebble problem will also be useful for algorithmic discussions, as in the proof of Theorem 2. From now on, this section does not refer anymore to the question of assignments and periodic tasks.

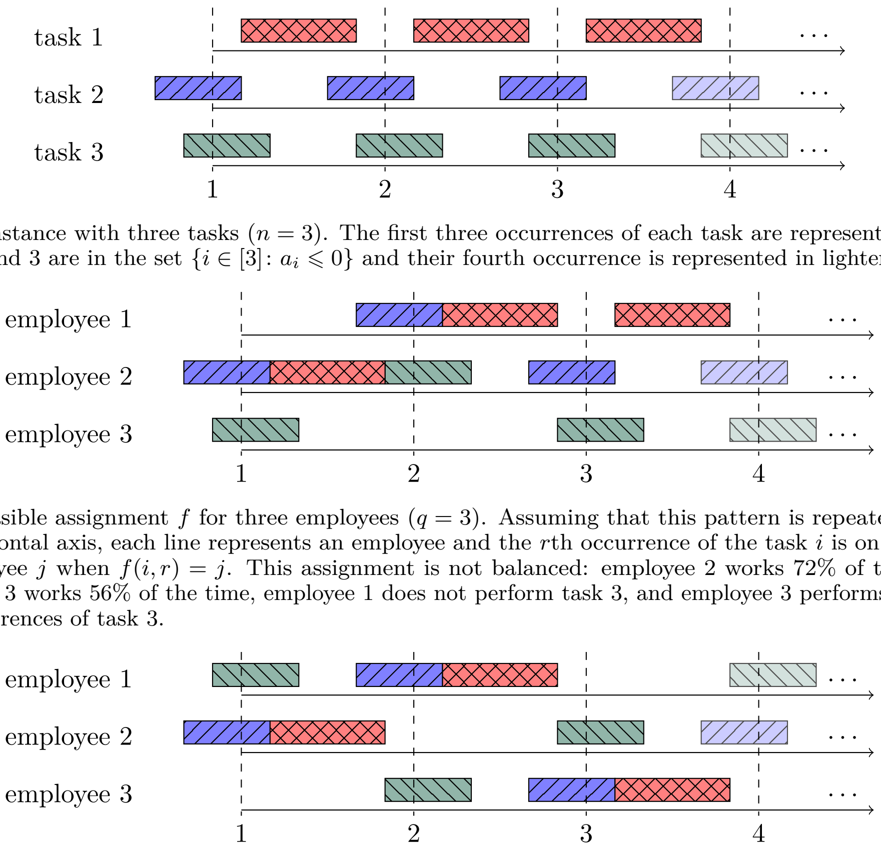
Consider an arc-colored Eulerian directed multi-graph $D = (V, A)$ such that each vertex is the head of exactly one arc of every color, and also the tail of exactly one arc of every color. (In other words, each color is a collection of vertex disjoint directed cycles covering the vertex set.) Assume we have a pebble on each vertex. We denote by P the set of pebbles, and we have thus $|P| = |V|$.

Now, we explain how a sequence of colors induces a sequence of moves for the pebbles. Given a sequence $c_1, c_2, \ldots$ of colors, each pebble is first moved along the unique arc of color c_1 leaving the vertex on which it is originally located; then it is moved along the unique arc of color c_2 leaving the vertex it has reached after the first move; and so on. Remark that each move sends each pebble on a distinct vertex and so after each move, there is again a pebble on each vertex.

We might ask under which condition there exists an infinite sequence of colors such that the arc visits are "balanced," i.e., each pebble visits each arc with the same frequency. Not only such a sequence always exists but such a sequence can be chosen to be periodic.

▶ **Proposition 3.** *There always exists a periodic sequence of colors making each pebble visit each arc with the same frequency.*

The proof shows a bit more: each pebble actually follows a periodic walk on D which has the same period as the sequence of colors, and the latter is upper bounded by $|A|(|V| - 1)!$.



(a) An instance with three tasks ($n = 3$). The first three occurrences of each task are represented. The tasks 2 and 3 are in the set $\{i \in [3] : a_i \leqslant 0\}$ and their fourth occurrence is represented in lighter color.

(b) A feasible assignment f for three employees ($q = 3$). Assuming that this pattern is repeated along the horizontal axis, each line represents an employee and the rth occurrence of the task i is on the line of employee j when $f(i, r) = j$. This assignment is not balanced: employee 2 works 72% of the time, employee 3 works 56% of the time, employee 1 does not perform task 3, and employee 3 performs 75% of the occurrences of task 3.

(c) A feasible assignment g for three employees ($q = 3$). Assuming that this pattern is repeated along the horizontal axis, the assignment g is feasible and balanced: task 1 is performed equally by employees 1, 2, and 3, and so are tasks 2 and 3. The assignment g is periodic with period $h = 3$.

Figure 1 Example of an instance, with two feasible assignments.

The proof of this proposition relies on a larger graph $\widetilde{D} = (\widetilde{V}, \widetilde{A})$ built as follows. The vertex set $\widetilde{V}$ is the set of bijections from P to V. For every color c, define the permutation σ^c of V by setting $\sigma^c(i) = i'$ whenever there is an arc of color c from i to i' in D. The set $\widetilde{A}$ is built as follows: for each bijection $\eta \colon P \to V$ and each color c, introduce an arc $(\eta, \sigma^c \circ \eta)$, and color this arc with color c. The indegree and outdegree of every vertex in $\widetilde{V}$ are equal to the number of colors.

For each pebble j, we introduce a function $p_j \colon \widetilde{A} \to A$. Given an arc $\tilde{a} = (\eta, \eta')$ of $\widetilde{A}$ with color c, we define $p_j(\tilde{a})$ as the arc $(\eta(j), \eta'(j))$ of A with color c.

The graph $\widetilde{D}$ is an encoding of all possible distributions of the pebbles on V and all possible transitions between these distributions. More precisely consider any initial distribution η of the pebbles on V and a sequence of colors $c_1, c_2, \ldots$. The moves induced by the sequence of colors translate into a walk on $\widetilde{D}$. The corresponding sequence of vertices of $\widetilde{D}$ is the sequence of distributions of the pebbles on V induced by the sequence of colors.

▶ **Lemma 4.** *Let $j \in P$, $a \in A$, and $\widetilde{K}$ be a connected component of $\widetilde{D}$ (note that weakly and strongly connected components of $\widetilde{D}$ are identical by equality of the in- and outdegrees). Denoting by κ the number of connected components of $\widetilde{D}$, we have*

$$|p_j^{-1}(a) \cap A(\widetilde{K})| = \frac{(|V| - 1)!}{\kappa}\,.$$

In particular, the left-hand term is independent of j, a, and $\widetilde{K}$.

Proof. Denote by c the color of a.

We prove first that every connected component $\widetilde{K}$ of $\widetilde{D}$ contains at least one arc from $p_j^{-1}(a)$. Let η be a vertex of such a connected component $\widetilde{K}$. Consider any walk W in D from $\eta(j)$ to the tail of a, and then traversing a. Such a walk exists because D is strongly connected. With $c_1, c_2, \ldots, c$ being the sequence of colors of the arcs traversed by the walk, the sequence

$$\eta, \sigma^{c_1} \circ \eta, \sigma^{c_2} \circ \sigma^{c_1} \circ \eta, \ldots, \sigma^{c} \circ \cdots \circ \sigma^{c_2} \circ \sigma^{c_1} \circ \eta$$

forms a walk in $\widetilde{K}$ starting from η, whose image by p_j is W. Hence, $\widetilde{K}$ contains at least one arc from $p_j^{-1}(a)$.

Second, given two components $\widetilde{K}_1$ and $\widetilde{K}_2$ of $\widetilde{D}$, we build an injective map $\psi \colon A(\widetilde{K}_1) \to A(\widetilde{K}_2)$ as follows (actually, it is a bijection but this property is not explicitly used). Pick $\tilde{a}_1 \in p_j^{-1}(a) \cap A(\widetilde{K}_1)$ and $\tilde{a}_2 \in p_j^{-1}(a) \cap A(\widetilde{K}_2)$. According to what we have just proved, these two arcs exist. Write $\tilde{a}_1 = (\eta_1, \sigma^c \circ \eta_1)$ and $\tilde{a}_2 = (\eta_2, \sigma^c \circ \eta_2)$. Then, for an arc $\tilde{a} \in A(\widetilde{K}_1)$ with tail vertex η and color d, set $\psi(\tilde{a})$ as the arc $(\eta \circ \eta_1^{-1} \circ \eta_2, \sigma^d \circ \eta \circ \eta_1^{-1} \circ \eta_2)$ with color d (this arc is unique). Checking that ψ is injective is immediate.

Third, we check that ψ maps elements from $p_j^{-1}(a) \cap A(\widetilde{K}_1)$ to $p_j^{-1}(a) \cap A(\widetilde{K}_2)$. Let $\tilde{a}$ be an arc in $p_j^{-1}(a) \cap A(\widetilde{K}_1)$. It is of the form $(\eta, \sigma^c \circ \eta)$. Its image by ψ is the arc $(\eta \circ \eta_1^{-1} \circ \eta_2, \sigma^c \circ \eta \circ \eta_1^{-1} \circ \eta_2)$ with color c. Denoting i the tail of a, we have $\eta(j) = \eta_1(j) = \eta_2(j) = i$, which implies immediately that $p_j\big(\psi(\tilde{a})\big)$ has the same endpoints as a. Since it has also the same color c, we have $p_j\big(\psi(\tilde{a})\big) = a$.

From the previous two paragraphs, we see that for any two components $\widetilde{K}_1$ and $\widetilde{K}_2$ of $\widetilde{D}$, we have $|p_j^{-1}(a) \cap A(\widetilde{K}_1)| \leqslant |p_j^{-1}(a) \cap A(\widetilde{K}_2)|$. Since the choices of $\widetilde{K}_1$ and $\widetilde{K}_2$ can be arbitrary, we have actually

$$|p_j^{-1}(a) \cap A(\widetilde{K}_1)| = |p_j^{-1}(a) \cap A(\widetilde{K}_2)| \,. \tag{3}$$

Finally, an arc $\tilde{a} = (\eta, \eta')$ is mapped to a by p_j precisely when $\tilde{a}$ is colored with color c, we have $\eta' = \sigma^c \circ \eta$, and $\eta(j) = i$ (where i is the tail of a). The number of bijections η from P to V with $\eta(j) = i$ is $(|V| - 1)!$. Hence, $|p_j^{-1}(a)| = (|V| - 1)!$. Combining this with equality (3), we get the desired conclusion. ◀

Proof of Proposition 3. Choose any connected component $\widetilde{K}$ of $\widetilde{D}$. It is Eulerian, since each vertex of $\widetilde{D}$ has equal in- and outdegrees. Consider an arbitrary Eulerian cycle, and denote by $c_1, c_2, \ldots$ the sequence of colors of the arcs of this cycle. According to Lemma 4, every pebble j moved according to this sequence of colors follows a closed walk on D visiting each arc $\frac{(|V|-1)!}{\kappa}$ times. Repeating infinitely many times this sequence of colors provides the desired periodic sequence. ◀

3 Proofs of the main results

3.1 Preliminaries

This section introduces preliminary results and a few tools that will be crucial for the proofs of Theorems 1 and 2 given in Section 3.2. In particular, we show how to introduce fictitious tasks in a way that will simplify some discussions, we explain how to build a new feasible assignment from a sequence of feasible assignments, and finally we define a graph $D^{\mathcal{F}}$ built from a set of feasible assignments, which will be useful to cast the problem of existence of balanced feasible assignments as the pure graph problem of Section 2.

3.1.1 Making the number of employees and the number of tasks overlapping 0 equal

Denote by U the set $\{i \in [n] \colon a_i \leqslant 0\}$. In other words U is the set of tasks that are overlapping the left endpoint of the interval $[0,1]$. We can get $|U| = q$ by adding fictitious tasks i whose intervals are of the form $[0, \varepsilon)$ for an $\varepsilon > 0$ small enough. Just after the proof of the following lemma, the relevance of this transformation will be highlighted.

▶ **Lemma 5.** *The number ε can be chosen so that the following holds: There exists a feasible assignment for the original instance if and only if there exists a feasible assignment for the instance with the fictitious tasks.*

Proof. Let $\varepsilon := \min\big(\{a_i \colon a_i > 0\} \cup \{a_i + 1 \colon a_i \leqslant 0\}\big)$. Obviously, if there exists a feasible assignment with the fictitious tasks, then the restriction of this assignment to the original tasks is feasible. Conversely, suppose there exists a feasible assignment f for the original instance. We extend it on the fictitious tasks as follows: for each integer time, the assignment f leaves a number of idle employees equal to the number of fictitious tasks; extending f arbitrarily on the rth occurrence of these tasks with these employees leads to a feasible assignment for the instance with the fictitious tasks. (The number ε has been chosen so that this does not create any conflict.) ◀

Since a balanced feasible assignment for the instance with the fictitious tasks is obviously balanced and feasible when restricted to the original instance, the assumption $|U| = q$ is made throughout Section 3. For every feasible assignment f and every $r \in \mathbb{Z}_{>0}$, we introduce the map $\varphi_{f,r} \colon i \in U \mapsto f(i,r) \in [q]$. The assumption $|U| = q$ makes $\varphi_{f,r}$ a bijection, which will turn out to be useful, already in the next paragraph.

3.1.2 Building a new feasible assignment from a sequence of feasible assignments

In the proofs, we will build new feasible assignments from sequences of feasible assignments. Let $f_1, f_2, \ldots$ be an infinite sequence of feasible assignments. Define inductively the permutations π_r of $[q]$ by the equation $\pi_{r+1} = \pi_r \circ \varphi_{f_r,2} \circ \varphi^{-1}_{f_{r+1},1}$, where π_1 is an arbitrary permutation of $[q]$. This implies in particular

$$(\pi_{r+1} \circ f_{r+1})(\cdot, 1) = (\pi_r \circ f_r)(\cdot, 2)\,. \tag{4}$$

Notice that if the f_r are periodic, then so are the π_r. Note also that the period of the π_r can be much larger than that of the f_r. (Similar constructions have been used in the work of Eisenbeis, Lelait, and Marmol [3].)

▶ **Lemma 6.** *The map $(i,r) \mapsto (\pi_r \circ f_r)(i,1)$ is a feasible assignment.*

Proof. Let us show that $g \colon (i,r) \mapsto (\pi_r \circ f_r)(i,1)$ is a feasible assignment by checking the contrapositive of (1). Consider $i, i' \in [n]$ with $i \neq i'$ and $r, r' \in \mathbb{Z}_{>0}$. Suppose $g(i,r) = g(i',r')$, i.e., $(\pi_r \circ f_r)(i,1) = (\pi_{r'} \circ f_{r'})(i',1)$. Without loss of generality, suppose that $r \leqslant r'$. Consider first the case when $r = r'$. Since π_r is a permutation, we have $f_r(i,1) = f_r(i',1)$. Since f_r is feasible, then the contrapositive holds for f_r, namely, $[a_i + 1, b_i + 1) \cap [a_{i'} + 1, b_{i'} + 1) = \varnothing$, which is equivalent to $[a_i + r, b_i + r) \cap [a_{i'} + r', b_{i'} + r') = \varnothing$, as desired.

Consider now the case when $r + 1 = r'$. Note first that if $i' \notin U$, then $b_i + r < a_{i'} + r + 1$ and so $[a_i + r, b_i + r) \cap [a_{i'} + r', b_{i'} + r') = \varnothing$. Suppose now that $i' \in U$. Using the definition of π_{r+1}, we have $\pi_r(f_r(i,1)) = \pi_r(f_r(i',2))$. Since π_r is a permutation, then $f_r(i,1) = f_r(i',2)$. The contrapositive holds for the feasible assignment f_r, so $[a_i + r, b_i + r) \cap [a_{i'} + r', b_{i'} + r') = \varnothing$.

Finally, if $r + 2 \leqslant r'$, then $[a_i + r, b_i + r) \cap [a_{i'} + r', b_{i'} + r')$ is necessarily empty. Therefore, g is a feasible assignment. ◀

3.1.3 Definition of $D^{\mathcal{F}}$

For each $i, i' \in U$, pick a feasible assignment $f_{ii'}$ such that $f_{ii'}(i,1) = f_{ii'}(i',2)$ if it exists. Let $\mathcal{F}$ be the set of all these feasible assignments (some $f_{ii'}$ might be equal but only one representative is kept). The proofs rely on a directed multi-graph $D^{\mathcal{F}} = (U, A^{\mathcal{F}})$, with vertex set U and whose arcs are defined from the set $\mathcal{F}$. The arc set $A^{\mathcal{F}}$ is obtained by introducing q arcs for each $f \in \mathcal{F}$: an arc from $i \in U$ to $i' \in U$ whenever $f(i,1) = f(i',2)$ – repetitions are allowed – ; such an arc is labeled with f.

Note the following properties:

- The q arcs labeled with the same feasible assignment f form a collection of vertex-disjoint directed cycles: each vertex is by construction the head of exactly one arc and the tail of exactly one arc.
- The number of arcs in $A^{\mathcal{F}}$ is $q|\mathcal{F}|$.
- When $D^{\mathcal{F}}$ is weakly connected – i.e., the underlying undirected graph is connected – it is also strongly connected and Eulerian.

Lemma 6 will be used to retrieve a feasible assignment from a walk on $D^{\mathcal{F}}$. The next lemma will be useful in that regard.

▶ **Lemma 7.** *The directed graph $D^{\mathcal{F}}$ is Eulerian if and only if there exists a feasible assignment with an employee performing each task at least once.*

Proof. Suppose there exists a feasible assignment f with an employee $j^\star$ performing each task at least once. For every r such that $f(i,r) = j^\star$ and $f(i', r+1) = j^\star$ there is an arc (i,i') in $A^{\mathcal{F}}$ because we can build a feasible assignment from f in which the first occurrence of i and the second occurrence of i' are both assigned to employee $j^\star$. Thus the sequence of tasks in U performed by employee $j^\star$ induces in $D^{\mathcal{F}}$ a walk visiting all vertices. This implies that the graph $D^{\mathcal{F}}$ is weakly connected and as noted above this implies that $D^{\mathcal{F}}$ is Eulerian.

Suppose now that $D^{\mathcal{F}}$ is Eulerian. Let $a_1, a_2, \ldots$ be the sequence of arcs of an Eulerian cycle of $D^{\mathcal{F}}$, visited infinitely many times, and consider the sequence $f_1, f_2, \ldots$ of assignments labeling this arc sequence. Denote by i_r the tail of the arc a_r. Let π_r be the permutations defined by equation (4), for this sequence of assignments, with π_1 being arbitrary. Let then g be the feasible assignment as in Lemma 6. Let $i \in [n]$. Define then $i' := \varphi^{-1}_{f_1,1}(f_1(i,1))$. In particular, we have $f_1(i,1) = f_1(i',1)$. In other words, i' is the task in U performed before the first occurrence of i in the assignment f_1. Let $\bar{r}$ be such that $a_{\bar{r}}$ leaves the vertex i' with label f_1, which means that $i' = i_{\bar{r}}$ and $f_{\bar{r}} = f_1$. Such $\bar{r}$ exists because an Eulerian cycle visits all arcs and because from every vertex, there is an outgoing arc labeled with f_1 by construction of $D^{\mathcal{F}}$. Thus, we have

$$g(i,\bar{r}) = \pi_{\bar{r}}(f_{\bar{r}}(i,1)) = \pi_{\bar{r}}(f_1(i,1)) = \pi_{\bar{r}}(f_1(i_{\bar{r}},1)) = \pi_{\bar{r}}(f_{\bar{r}}(i_{\bar{r}},1))\,.$$

We have $f_r(i_r,1) = f_r(i_{r+1},2)$ for all $r \in \mathbb{Z}_{>0}$ because the arc a_r is labeled with f_r and a_r goes from i_r to i_{r+1}. Then, combining this equality alternatively with the equation (4), we get

$$\pi_{\bar{r}}(f_{\bar{r}}(i_{\bar{r}},1)) = \pi_{\bar{r}-1}(f_{\bar{r}-1}(i_{\bar{r}},2)) = \pi_{\bar{r}-1}(f_{\bar{r}-1}(i_{\bar{r}-1},1)) = \cdots = \pi_1(f_1(i_1,1)) = g(i_1,1)\,.$$

Therefore $g(i,\bar{r}) = g(i_1,1)$. Since i was chosen arbitrarily, this means the employee $g(i_1,1)$ performs each task at least once. ◀

3.2 Proof of Theorems 1 and 2

This section deals with the proofs of our two main results.

Proof of Theorem 1. As already mentioned, one direction is immediate: if there exists a balanced feasible assignment, then this assignment is such that every employee performs each task at least once. We prove now the opposite direction.

Suppose there exists a feasible assignment with an employee performing each task at least once. By Lemma 7, $D^{\mathcal{F}}$ is Eulerian. Locate one pebble on each vertex of $D^{\mathcal{F}}$. Applying Proposition 3 on $D^{\mathcal{F}}$, with each feasible assignment in $\mathcal{F}$ identified with a color, we get a periodic sequence $f_1, f_2, \ldots$ of feasible assignments. Denote by g the resulting periodic feasible assignment given by Lemma 6 for this sequence, with π_1 being an arbitrary permutation.

Number $j = g(i, 1)$ the pebble initially located on vertex $i \in U$. This makes sure that each pebble gets a distinct number in $[q]$ (by the bijectivity of $\varphi_{g,1}$). We establish now the following claim: *For every $i \in U$ and every $j \in [q]$, pebble j is on vertex i after its $(r-1)$th move if and only if $g(i, r) = j$.*

Let us proceed by induction on $r \in \mathbb{Z}_{>0}$. This is true for $r = 1$ by the definition of the numbering of the pebbles. Suppose now that the claim if true for some $r \in \mathbb{Z}_{>0}$. Consider pebble j and assume it is located on i after its rth move. This means that the pebble j was on vertex i' after its $(r-1)$th move then moved along the arc from i' to i labeled with f_r. Then, using equation (4) and the fact that $f_r(i, 2) = f_r(i', 1)$ by definition of $D^{\mathcal{F}}$,

$$g(i, r+1) = \pi_{r+1}(f_{r+1}(i, 1)) = \pi_r(f_r(i, 2)) = \pi_r(f_r(i', 1)) = g(i', r) = j\,, \tag{5}$$

as desired. Conversely, assume that $g(i, r+1) = j$. Denote by i' the tail of the arc of head i and label f_r. Then equation (5) holds as well, meaning that $g(i', r) = j$. By induction, the pebble j was located on vertex i' after its $(r-1)$th move. It then moves along the arc from i' to i with label f_r, which concludes the proof of the claim.

We check that g is balanced. According to Proposition 3, for every $i \in U$, every $j \in [q]$, and every $f \in \mathcal{F}$, we have

$$\lim_{t \to +\infty} \frac{1}{t} \big|\{r \in [t] \colon \text{pebble } j \text{ leaves } i \text{ along arc labeled } f \text{ for its } (r-1)\text{th move}\}\big| = \frac{1}{|A^{\mathcal{F}}|}\,.$$

With the claim, this equality becomes

$$\lim_{t \to +\infty} \frac{1}{t} \big|\{r \in [t] \colon f_r = f \text{ and } g(i, r) = j\}\big| = \frac{1}{|A^{\mathcal{F}}|}\,.$$

This equality is actually also true when U is replaced by the larger set $[n]$. Indeed, given $i \in [n]$ and $f \in \mathcal{F}$, the bijectivity of $\varphi_{f,r}$ ensures that there exists a unique $u(i, f)$ in U such that $f(u(i, f), r) = f(i, r)$ and we have $g(i, r) = g(u(i, f_r), r) = j$ for every $r \in \mathbb{Z}_{>0}$, by definition of g and $u(i, f_r)$. Therefore, for all $i \in [n]$

$$\lim_{t \to +\infty} \frac{1}{t} \big|\{r \in [t] \colon g(i, r) = j\}\big| = \sum_{f \in \mathcal{F}} \lim_{t \to +\infty} \frac{1}{t} \big|\{r \in [t] \colon f_r = f \text{ and } g(i, r) = j\}\big| = \frac{|\mathcal{F}|}{|A^{\mathcal{F}}|} = \frac{1}{q}\,,$$

as desired. ◀

The proof actually shows that the period of the periodic balanced feasible assignment g built within the proof is upper bounded by $q^2 \times q!$. Indeed, with the comment following Proposition 3, each vertex i is visited by each pebble every h moves (with h the period of the f_r), where h is bounded by $|A^{\mathcal{F}}|(q-1)! \leqslant q^2 \times q!$. Using the claim in the proof of Theorem 1, for each $i \in U$ and $r \in \mathbb{Z}_{>0}$, we have $g(i,r) = g(i,r+h)$. Extending this relation to all $i \in [n]$ as in the proof of Theorem 1 (since the sequence of the f_r also has period h), the period of g is bounded by $q^2 \times q!$.

The proof of Theorem 2 combines this remark, Theorem 1, Lemma 7, and the following lemma.

▶ **Lemma 8.** *Let i, i' be two tasks of U. Deciding whether there exists a feasible assignment f such that $f(i,1) = f(i',2)$ and building such an assignment if it exists can be done in polynomial time.*

Proof. Let G be the interval graph built from all the intervals $[a_k, b_k)$ for $k \in [n]$ together with the intervals $[a_k\!+\!1, b_k\!+\!1)$ for $k \in U$. Deciding whether there exists a feasible assignment f such that $f(i,1) = f(i',2)$ is equivalent to deciding whether there is a proper q-coloring of G with the intervals $[a_i, b_i)$ and $[a_{i'}\!+\!1, b_{i'}\!+\!1)$ colored the same color. (Indeed, such a feasible assignment translates into a proper q-coloring of G with the desired property and conversely such a q-coloring provides a "partial" feasible assignment which can be extended into a feasible one easily.) This is equivalent in turn to the problem of deciding the q-colorability of G with $[a_i, b_i)$ and all intervals intersecting $[a_{i'} + 1, b_{i'} + 1)$ (the latter interval excluded) colored with pairwise distinct colors. Here, we use the fact that $|U| = q$, i.e., there are $q - 1$ intervals intersecting $[a_{i'} + 1, b_{i'} + 1)$. The problem of deciding whether a partial coloring of an interval graph can be extended to a proper q-coloring can be done in polynomial time when the partial coloring contains at most one occurrence of each color (this is a result by Biró, Hujter, and Tuza [1]). If such a partial coloring extension exists, then it can be built in polynomial time as well. ◀

Proof of Theorem 2. According to Lemma 8, the graph $D^{\mathcal{F}}$ can be built in polynomial time (since $|\mathcal{F}| \leqslant q^2$). Deciding whether a graph is strongly connected can be done in polynomial time, and $D^{\mathcal{F}}$ being strongly connected means it is Eulerian. Therefore, using Theorem 1 along with Lemma 7, deciding whether there exists a balanced feasible assignment can be done in polynomial time.

Moreover, Theorem 1 provides a construction of a periodic balanced feasible assignment g (if it exists). By expliciting the arguments, the construction consists first in building the graph $\widetilde{D}$ of Lemma 4, and then in computing an Eulerian cycle in an arbitrary connected component of this graph (as done in the proof of Proposition 3), which provides a periodic sequence of feasible assignments $f_1, f_2, \ldots$. The size of $\widetilde{D}$ and the period of this sequence are both polynomial when q is fixed. In other words, the sequence can be described in polynomial time when q is fixed. This allows a polynomial description of g when q is fixed according to the comment following the proof of Theorem 1. ◀

4 Concluding remarks

4.1 All feasible assignments are balanced (Almost)

If we are just interested in the existence of a balanced feasible assignment, and not on the periodicity of such an assignment or its computability, we can replace Proposition 3 by the following lemma in the proof of Theorem 1. We keep the same setting of an arc-colored Eulerian directed multi-graph $D = (V, A)$ with a distribution of pebbles on its vertices, as in the beginning of Section 3.2.

▶ **Lemma 9.** *Consider an infinite sequence of independent random colors drawn uniformly. Then, almost surely, this sequence makes each pebble visit each arc with the same frequency.*

In particular there are infinitely many color sequences making each pebble visit each arc with the same frequency. The proof relies on basic properties of Markov chains. (A standard reference on Markov chains is the book by Norris [6].) The proof does not show how to construct such a sequence of colors. It is not even clear that the proof could be modified in that regard. So the proof shows that almost all color sequences have the desired property, but does not explain how to construct a single such sequence. Although this might sound surprising, this phenomenon is quite common. *Normal* numbers form an example: (almost) all numbers are normal but not a single one has been described explicitely [9].

The proof of Proposition 3 actually provides an alternative proof of the existence of sequences of colors making each pebble visit each arc with the same frequency, with an explicit construction. However, the latter proof does not show that almost all sequences are actually like that. (In counterpart, it shows that such a sequence can be chosen to be periodic.)

Proof of Lemma 9. Any realization of this random sequence of colors defines a sequence of moves for the pebbles, as described above. Consider an arbitrary pebble. The random sequence of colors translates thus into a random walk of the pebble on the graph D. Denote by X_k the arc along which the pebble performs its kth move. The X_k's form a finite Markov chain. Since the graph is Eulerian, this Markov chain is irreducible, and hence there exists a unique invariant distribution λ such that $\lambda^\top = \lambda^\top M$, where M is the transition matrix of the Markov chain.

We claim that λ is actually the vector $\frac{1}{|A|}e$, where e is the all-one vector. By the uniqueness of the invariant distribution, it is enough to check that e is a left eigenvector of M with eigenvalue equal to 1. The entry $M_{a,a'}$ of the transition matrix (row a, column a'), which corresponds to the probability of moving along a' just after moving along a, is equal to $1/\alpha$ if the head of a is the tail of a', and 0 otherwise. The indegree of each vertex being α, we have

$$\sum_{a\in A} M_{a,a'} = \alpha\frac{1}{\alpha} = 1\,,$$

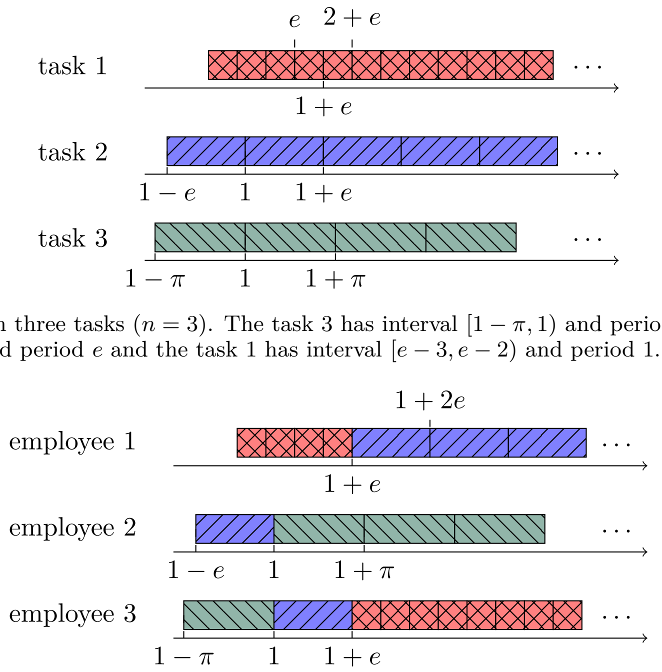
and therefore $e^\top M = e^\top$.

According to the ergodic theorem, for almost all realizations of the random sequence of colors, the pebble visits any arc a with a frequency equal to the corresponding entry in λ, which is equal to $\frac{1}{|A|}$ for all arcs since it is a probability distribution proportional to the all-one vector. The previous discussion does not depend on the considered pebble, which implies the desired result: for almost all realizations of the random sequence of colors, every pebble visits every arc with a frequency equal to $\frac{1}{|A|}$. ◀

Similarly to the proof of Theorem 1, using a much larger set of feasible assignments $\mathcal{F}'$ (typically one for each feasible "pattern" on $[1,2]$), Lemma 9 could translate into the following statement: *If there is at least one balanced feasible assignment, then almost all feasible assignments are balanced.*

4.2 Tasks with different periods

Suppose now that each task $i \in [n]$ comes with a period $\tau_i \geqslant b_i - a_i$: the rth occurrence of task i takes place over the time interval $[a_i + r\tau_i, b_i + r\tau_i)$. In this more general setting, it is not clear under which extra condition Theorem 1 and Theorem 2 are verified. When some periods are irrational, the equivalence stated by Theorem 1 does not necessarily hold. Figure 2 provides an example where there exists a feasible assignment with one employee performing all the tasks at least once but no feasible assignment is balanced.



(a) An instance with three tasks ($n = 3$). The task 3 has interval $[1-\pi, 1)$ and period π, the task 2 has interval $[1-e, 1)$ and period e and the task 1 has interval $[e-3, e-2)$ and period 1.

(b) A feasible assignment f for three employees ($q = 3$). Assuming that this pattern is extended along the horizontal axis, the employee 3 performs each task at least once.

Figure 2 Example of an instance with irrational periods, for $q = 3$, with a feasible assignment where an employee performs each task at least once but with no balanced feasible assignment: after time $1 + e$, there is no possible swap of the tasks between the employees.

However, the authors do not know what happens when all the periods are rational. In the latter case, there is a natural way to associate to the original instance a new instance, with all tasks of period 1 and with the same number of employees. This goes as follows. Write the τ_i as fractions of integers with the same denominator, and denote by p_i the numerator. Let p be a common multiplier of the p_i. For every $i \in [n]$ interpret the p/p_i first occurrences of tasks i as the first occurrence of p/p_i new tasks that replace the original task i. Then, scale the time with a factor $1/p$ so as to get a common period of 1. Clearly, a balanced assignment for this new instance translates into a balanced one for the original instance, but it is not clear whether things go the other way around. Moreover, the construction described above is not polynomial and extending Theorem 2 along these lines seems even more elusive.

4.3 When the number of employees is part of the instance

Theorem 2 states that when the number of employees q is constant, a balanced feasible assignment (if it exists) can be built in polynomial time. However, the proof exhibits a period bounded by $q^2 \times q!$ for this balanced feasible assignment. Therefore, when q is part of the instance, it is not clear whether the construction of a balanced feasible assignment (under condition of existence) is polynomial.

▶ **Question.** *When the number q of employees is part of the instance, what is the complexity status of the construction of a balanced feasible assignment (if it exists)?*

References

1 Miklós Biró, Mihály Hujter, and Zsolt Tuza. Precoloring extension. I. Interval graphs. *Discrete Mathematics*, 100(1-3):267–279, 1992.

2 Thomas Breugem, Twan Dollevoet, and Dennis Huisman. Analyzing a family of formulations for cyclic crew rostering. In Dennis Huisman and Christos D Zaroliagis, editors, *ATMOS 2020*, volume 85, pages 7:1–7:16, 2020.

3 Christine Eisenbeis, Sylvain Lelait, and Bruno Marmol. Circular-arc graph coloring and unrolling, February 1998.

4 Jan Korst, Emile Aarts, Jan Karel Lenstra, and Jaap Wessels. Periodic assignment and graph colouring. *Discrete Applied Mathematics*, 51(3):291–305, 1994.

5 Christian Liebchen and Rolf Möhring. The modeling power of the periodic event scheduling problem: Railway timetables – and beyond. In *Computer-aided Systems in Public Transport*, volume 600, pages 117–150. Springer Berlin Heidelberg, January 2008.

6 James R Norris. *Markov chains*. Cambridge university press, 1998.

7 Philine Schiewe and Anita Schöbel. Periodic timetabling with integrated routing: Toward applicable approaches. *Transportation Science*, 54(6):1714–1731, 2020.

8 Paolo Serafini and Walter Ukovich. A mathematical model for periodic scheduling problems. *SIAM Journal on Discrete Mathematics*, 2(4):550–581, 1989.

9 Wikipedia. Normal number, 2024. [Online; accessed 1-July-2024]. URL: https://en.wikipedia.org/wiki/Normal_number.

10 Lin Xie and Leena Suhl. Cyclic and non-cyclic crew rostering problems in public bus transit. *OR Spectrum*, 37:99–136, 2015.

Two-Stage Weekly Shift Scheduling for Train Dispatchers

Tomas Lidén
Department of Science and Technology (ITN), Linköping University, Sweden

Christiane Schmidt
Department of Science and Technology (ITN), Linköping University, Sweden

Rabii Zahir
Department of Science and Technology (ITN), Linköping University, Sweden

Abstract

We consider the problem of creating weekly shift schedules for train dispatchers, which conform to a variety of operational constraints, in particular, several work and rest time restrictions. We create the schedules in a two-stage process. First, using a previously presented IP model, we create a set of feasible daily shifts, which takes care of minimum-rest and shift-length requirements, taskload bounds, and combinability of dispatching areas. We then formulate an IP model to combine these daily shifts into weekly schedules, enforcing that each daily shift is covered by some dispatcher every day of the week, while ensuring that the weekly schedules comply with various restrictions on working hours from a union agreement. With this approach, we aim to identify "good" sets of daily shifts for the longer schedules. We run experiments for real-world sized input and consider different distributions of the daily shifts w.r.t. shift length and ratio of night shifts. Daily shifts with shift-length variability, relatively few long shifts, and a low ratio of night shifts generally yield better weekly schedules. The runtime for the second stage with the best daily-shift pattern is below three hours, which – together with the runtime for stage 1 of ca. 2 hours per run – can be feasible for real-world use.

2012 ACM Subject Classification Applied computing → Transportation

Keywords and phrases shift scheduling, IP, train dispatcher shift scheduling

Digital Object Identifier 10.4230/OASIcs.ATMOS.2024.6

Funding This work is part of the Trafikverket-funded project Capacity Modeling and Shift Optimization for Train Dispatchers (CAPMO-Train) (TRV 2020/99267). Partially funded by the Swedish Research Council through grant agreement no. 2022-06725.

Acknowledgements The computations were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS) at Linköping university (the Tetralith server). We would like to thank Magnus C. Johansson, Carl-Johan Wallentin, and Sebastian Olred from Trafikverket, TC Malmö, for their time and expertise in mapping the constraints on train-dispatcher shifts. Finally, we thank the reviewers of this paper for their valuable feedback.

1 Introduction

Train dispatchers are responsible for assigning tracks, for routing trains safely and efficiently, and for guaranteeing the safety of staff working close to the tracks [18]. A good performance in their dispatching work is crucial for safe and efficient operations. This becomes even more important with increasing traffic volumes: railway passenger traffic had increased pre-pandemic and has again caught up with these volumes [9].

In Sweden, the Swedish Transport Administration (Trafikverket) manages ca. 90% of the ∼15.600-km track network [21]. For this task, the network is divided into eight dispatching centers, each of which is further partitioned into geographical areas. Such a geographical area

24th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2024).
Editors: Paul C. Bouman and Spyros C. Kontogiannis; Article No. 6; pp. 6:1–6:16
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

must be controlled by a dispatcher, but when the traffic situation allows – in particular, with a low enough traffic volume – the working positions of several dispatchers, and with that the geographical areas, can be combined to fewer working positions. However, the dispatchers will need an endorsement for each area they control.

Apart from these operational requirements, dispatcher shifts must comply with a variety of restrictions from a union agreement [20]: limits on the length of a single shift, limits on the rest periods between consecutive shifts, limits on the weekly working hours (depending on whether and how much a dispatcher works during nights and weekends). Currently, shift scheduling for train dispatchers is a manual process and – up to our work – it had obtained little attention by researchers, despite its complexity. In previous work, we [16, 15, 14] presented integer programming (IP) models to compute daily train-dispatcher shifts, which are feasible w.r.t. all shift-specific legal and operational requirements and for which the dispatcher task load is limited.

While the daily shift plans are a good start, the planning horizon for dispatchers is longer. When moving from planning for a single day to planning for longer periods, for starters to a full week, a variety of relatively complicated restrictions from the union agreement [20] for Trafikverket come into play:

- The weekly rest in a seven-days period should include at least 36 hours of contiguous rest.
- In the average of the considered period (= the calendar year), ordinary working hours (refer to the standard weekly hours a dispatcher is expected to work excluding overtime), may not exceed:
  - On average 40 hours per week without holidays.
  - On average 38 hours per week for employees who have ordinary working hours before 06AM or after 8PM on average *one* time per week *or* ordinary working hours on a Saturday, Sunday, or holiday.
  - On average 36 hours per week for employees who have ordinary working hours before 6AM or after 8PM on average *two* times per week *and* ordinary working hours occur on Saturdays, Sundays, or holidays.
  - On average 36 hours per week in case of ordinary working hours which on average per employee occur 1 time/week *and* on average 2 hours/week and with on average 2 hours/week between 11PM and 5AM *and* ordinary working hours occur on Saturdays, Sundays, or holidays.
  - On average 34 hours and 15 minutes per week if ordinary working hours on average per employee occur with at least 1.4 times/week between 11PM and 5AM *and* at least once on average 2 hours between 11PM and 5AM *and* ordinary working hours occur on Saturdays, Sundays, and holidays.

We denote this list of restrictions by $\mathcal{R}$. Since the planned horizon considered is one week, resulting in a cyclic schedule, we use the total working hours to track the averages presented in $\mathcal{R}$.

Modeling all the constraints on weekly schedules mathematically incurs quite a high complexity, and we did not want to add these to the functionality of our models for daily shifts [16, 15, 14], as we do not expect to obtain solutions even with very high limits on the computation times. Hence, for this paper, we instead use the output of the optimization model for one-day shifts as puzzle bits that we aim to combine as good as possible for feasible weekly schedules. More precisely, we take a set of feasible one-day shifts as our puzzle bits and enforce that all of these are used every day – in contrast to using a larger set of possible shifts (puzzle bits), where our mathematical model would have to ensure that with the selected shifts each area is monitored by a dispatcher during all periods of each day (which would yield a column-generation approach). We use this approach, as we do not solely want to compute the optimal puzzle bits, but we aim to identify good puzzle bits (daily shifts) and how to create them. Thus, in our mathematical model for the weekly schedules, we do not need to take care of aspects covered in the daily shifts: dispatchers are assigned one or several combinable areas, for which they hold an endorsement; each dispatching area is assigned a dispatcher during every period of the day; each dispatcher shift complies with upper and lower bounds on the shift length; and the task load for each dispatcher does not exceed an upper bound in any time period. Of course with this approach we will likely not reach the global optimum for the weekly shifts. The runtime for stage one in our approach (using the previous models to compute daily shifts), even if we minimize the dispatcher-area-assignment switches (an operationally desirable property), is less than two hours.

In this paper, we minimize the number of dispatchers scheduled for work during the week. The motivation for this is twofold: the labor turnover for dispatchers is relatively high (possibly because of the partly undesirable working hours) and from the operational side it is interesting to know the minimal number of dispatchers that are needed to cover the existing traffic to be able to size the workforce; for us, the objective yields a baseline to which we can compare results for other possible objectives in later stages. This is in line with our work for daily shifts, where we first aimed to minimize the number of dispatchers and in later stages aimed to minimize the number of dispatcher-area assignment changes while keeping the number of dispatchers to the minimum.

Related Work. Shift-scheduling problems are typically different for each type of profession, and even for each specific case study, making them difficult to solve by standard models [19]. However, some tentative approaches for a generic shift scheduling were presented in [1, 19]. These approaches categorize the problems' constraints into hard ones, such as staffing and competence constraints, and soft ones (called horizontal) consisting of counter constraints (e.g., number of days off), series constraints (e.g., consecutive worked nights) and their combination (e.g., number of resting hours between consecutive shifts). The objective function contains a weighted sum of each of the violated horizontal constraints. In real life, where scheduling is done manually, some horizontal constraints may be violated [8]. In our paper, in contrast to [1, 19], we do not allow constraints to be violated, though we do have horizontal constraints, e.g., on consecutive hours off. Several solution approaches were proposed in the literature, which are based on exact methods, such as linear programming [4], constraint programming [13] and column generation [6, 7, 10, 12], or on heuristics, such as variable neighborhood search [3], tabu search [2, 17] and simulated annealing [5].

Guo et al., [11] used a two-stage approach for scheduling air traffic controllers over a two-week period. The type of shifts (morning, evening and night) and the corresponding demand were given. In the first step, they make shift-employee assignments, while breaks are added in the second phase. This paper is organized as follows: in Section 2, we describe the problem. In Section 3, we present the mathematical formulation. In Section 4, we describe the experiments and discuss corresponding results. Finally, in Section 5, we present our conclusions and recommendations for future work.

2 Problem Description

In this section, we briefly formalize the problem we aim to solve in this paper, the problem of shift scheduling for train dispatchers for several days:

Input: A set S of daily shifts that feasibly cover a single day; a set of train dispatchers, D; the length of the planning horizon in days, p; and a lower bound on the rest time between two consecutive shifts, $r^{\min}$.

Output: A dispatcher-minimal assignment of all daily shifts in S to dispatchers for each of the p days that fulfill all restrictions in $\mathcal{R}$, such that the rest time between consecutive shifts is at least $r^{\min}$.

3 Mathematical Formulation

In this section, we present an optimization model for the weekly schedules, with the parameters and variables presented in Tables 1 and 2, respectively. We give our model for daily shifts from [15, 14] in Appendix A for the ease of the reader.

The number of periods equals the number of days for which we are planning, that is, for a week, we have $p = 7$. The input is a set of shifts S that completely cover a day (where we for starters assume the same traffic volume during all 7 days) – S could contain an optimal set of shifts computed with one of our previous models [16, 15, 14], but also a different set of shifts (e.g., computed with the previous models but with other parameters on shift length, such as to obtain better "puzzle bits" for the weekly plan). We denote the elements of S as "daily shifts". We then construct the weekly schedule by enforcing that each shift is handled by some dispatcher every day.

We assume that each dispatcher holds endorsements for all areas, which provides a lower bound for other endorsement structures. If dispatchers hold endorsements only for some areas, we could define subsets $S_i \subseteq S$ of shifts that dispatcher i can cover.

We now give the constraints of our formulation, followed by the objective function and detailed explanations.

$$\sum_{i \in D} x_{i,j,k} = 1 \qquad \forall j \in S, \forall k \in P \tag{1}$$

$$\sum_{j \in S} x_{i,j,k} \leq 2 \qquad \forall i \in D, \forall k \in P \tag{2}$$

$$x_{i,j,k} \leq q_i \qquad \forall i \in D, \forall j \in S, \forall k \in P \tag{3}$$

$$\sum_{j \in S} \sum_{k \in P} x_{i,j,k} \geq q_i \qquad \forall i \in D \tag{4}$$

$$x_{i,j,k} + x_{i,j',k} \leq 1 \qquad \forall i \in D, \forall k \in P, \forall j, j' \in S : \Delta_{j,j'} < r^{\min} \tag{5}$$

$$x_{i,j,k} + x_{i,j',(k+1) \bmod p} \leq 1 \qquad \forall i \in D, \forall k \in P, \forall j, j' \in S : \Delta'_{j,j'} < r^{\min} \tag{6}$$

$$\sum_{k \in P} wr_{i,k} + wr'_{i,k} \geq q_i \qquad \forall i \in D \tag{7}$$

$$\delta_{j,j'} \cdot (x_{i,j,k} + x_{i,j',(k+1) \bmod p}) \leq 2 - wr_{i,k} \qquad \forall i \in D, \forall k \in P \quad \forall j, j' \in S \tag{8}$$

$$\delta'_{j,j'} \cdot (x_{i,j,k} + x_{i,j',(k+2) \bmod p}) \leq 2 - wr'_{i,k} \qquad \forall i \in D, \forall k \in P \quad \forall j, j' \in S \tag{9}$$

$$x_{i,j,(k+1) \bmod p} + wr'_{i,k} \leq 1 \qquad \forall i \in D, \forall j \in S, \forall k \in P \tag{10}$$

$$\sum_{j \in S} \sum_{k \in \{6,7\}} x_{i,j,k} + \sum_{\substack{j' \in S: \\ end_{j'} < start_{j'}}} x_{i,j',k'=5} \leq M \cdot wk_i \qquad \forall i \in D \tag{11}$$

$$\sum_{j \in S} \sum_{k \in \{6,7\}} x_{i,j,k} + \sum_{\substack{j' \in S: \\ end_{j'} < start_{j'}}} x_{i,j',k'=5} \geq wk_i \qquad \forall i \in D \tag{12}$$

$$\sum_{j\in S}\sum_{k\in P} o_j^{20/06}\cdot x_{i,j,k} + wk_i \quad \leq \quad M\cdot w_i^{38} \qquad \forall i\in D \tag{13}$$

$$\sum_{j\in S}\sum_{k\in P} o_j^{20/06}\cdot x_{i,j,k} + wk_i \quad \geq \quad w_i^{38} \qquad \forall i\in D \tag{14}$$

$$\sum_{j\in S}\sum_{k\in P} o_j^{20/06}\cdot x_{i,j,k} - 1 \quad \leq \quad M\cdot \theta_i^{20/06} \qquad \forall i\in D \tag{15}$$

$$\sum_{j\in S}\sum_{k\in P} o_j^{20/06}\cdot x_{i,j,k} \quad \geq \quad 2\cdot \theta_i^{20/06} \qquad \forall i\in D \tag{16}$$

$$\theta_i^{20/06} + wk_i - 1 \quad \leq \quad w_i^{36} \qquad \forall i\in D \tag{17}$$

$$\theta_i^{20/06} + wk_i \quad \geq \quad 2\cdot w_i^{36} \qquad \forall i\in D \tag{18}$$

$$\sum_{j\in S}\sum_{k\in P} \ell_j\cdot x_{i,j,k} + 2\cdot(w_i^{38} + w_i^{36}) \quad \leq \quad w^{\max} \qquad \forall i\in D \tag{19}$$

$$\sum_{j\in S}\sum_{k\in P} o'^{23/05}_j\cdot x_{i,j,k} - 1 \quad \leq \quad M\cdot \theta'^{23/05}_i \qquad \forall i\in D \tag{20}$$

$$\sum_{j\in S}\sum_{k\in P} o'^{23/05}_j\cdot x_{i,j,k} \quad \geq \quad 2\cdot \theta'^{23/05}_i \qquad \forall i\in D \tag{21}$$

$$\theta'^{23/05}_i + wk_i - 1 \quad \leq \quad w'^{36}_i \qquad \forall i\in D \tag{22}$$

$$\theta'^{23/05}_i + wk_i \quad \geq \quad 2\cdot w'^{36}_i \qquad \forall i\in D \tag{23}$$

$$\sum_{j\in S}\sum_{k\in P} \ell_j\cdot x_{i,j,k} + 4\cdot w'^{36}_i \quad \leq \quad w^{\max} \qquad \forall i\in D \tag{24}$$

$$\sum_{j\in S}\sum_{k\in P} o_j^{23/05}\cdot x_{i,j,k} - 1 \quad \leq \quad M\cdot \theta_i^{23/05} \qquad \forall i\in D \tag{25}$$

$$\sum_{j\in S}\sum_{k\in P} o_j^{23/05}\cdot x_{i,j,k} \quad \geq \quad 2\cdot \theta_i^{23/05} \qquad \forall i\in D \tag{26}$$

$$\theta_i^{23/05} + \theta'^{23/05}_i + wk_i \quad \leq \quad 2 + w_i^{34} \qquad \forall i\in D \tag{27}$$

$$\theta_i^{23/05} + \theta'^{23/05}_i + wk_i \quad \geq \quad 3\cdot w_i^{34} \qquad \forall i\in D \tag{28}$$

$$\sum_{j\in S}\sum_{k\in P} \ell_j\cdot x_{i,j,k} + 6\cdot w_i^{34} \quad \leq \quad w^{\max} \qquad \forall i\in D \tag{29}$$

The objective function minimizes the number of used dispatchers: $\min . \sum_{i\in D} q_i$ (30)

Constraint (1) ensures that each shift is assigned to exactly one dispatcher during each day. With Constraint (2), we limit the number of shifts a single dispatcher can be assigned during a day to two: with shifts of minimum length 6 and a rest period of at least 11 hours, that is the maximum possible number. Constraint (3) ensures that if a dispatcher is assigned to some shift during some day, they are counted as a working dispatcher, while Constraint (4) ensures that if a dispatcher is not used for any shift during any period, they are not counted as a working dispatcher. Constraint (5) states that if between shifts j and j', worked successively and started on the same day, we have less than the minimum resting time between two consecutive shifts, we can assign at most one of j and j' to the same dispatcher during a day. Similarly, Constraint (6) states that if between two shifts j and j', worked successively and j being an overnight shift, we have less than the minimum resting time between two consecutive shifts, we can assign at most one of j and j' to the same dispatcher during consecutive days. Constraint (7) ensures that any working dispatcher has at least one

Table 1 Model Parameters.

Parameter	Description
D	set of train dispatchers, indexed by i
S	set of daily shifts, indexed by j
P	set of week days in the time horizon, indexed by k
$start_j$	the beginning of the first working hour of shift j
end_j	the beginning of the last working hour of shift j
ℓ_j	the length of shift j
$\Delta_{j,j'}$	the gap (number of rest hours) between shifts j and j' if worked successively and start on the same day
$\Delta'_{j,j'}$	the gap (number of rest hours) between shifts j and j' if worked successively and the first shift ends the next day (overnight shift)
$\delta_{j,j'}$	binary, whether the gap between shifts j and j' is less than 36h if worked in two consecutive days
$\delta'_{j,j'}$	binary, whether the gap between shifts j and j' is less than 36h if started in k and $k+2$, respectively
$r^{\min}$	the minimum resting time between two consecutive shifts
$w^{\max}$	the maximum weekly working hours
$o_j^{20/06}$	binary, whether shift j overlaps period 20-06
$o_j^{23/05}$	binary, whether shift j overlaps period 23-05
$o'^{23/05}_j$	integer, the number of hours in shift j that overlap period 23-05
$p = \|P\|$	number of time periods (days) in the time horizon
$d = \|D\|$	number of available dispatchers (which maximally can be scheduled in the model)
M	a big number

Table 2 Model Variables.

Variable	Description
$x_{i,j,k}$	binary, whether disp. i is assigned shift j during period k
q_i	binary, whether disp. i works at least one shift/week
$wr_{i,k}$	binary, whether disp. i has weekly rest (36h) between periods k and $k+1$
$wr'_{i,k}$	binary, whether disp. i has weekly rest (36h) between periods k and $k+2$
wk_i	binary, whether disp. i works, at least once, on a weekend
$\theta_i^{20/06}$	binary, whether disp. i works at least twice in period 20-06
$\theta_i^{23/05}$	binary, whether disp. i works at least twice in period 23-05
$\theta_i'^{23/05}$	binary, whether disp. i works at least 2h/week in period 23-05
w_i^{38}	binary, whether disp. i works at least once in period 20-06 **or** on weekend
w_i^{36}	binary, whether disp. i works at least twice in period 20-06 **and** at least once on weekend
w'^{36}_i	binary, whether disp. i works at least 2h/week in period 23-05 **and** at least once on weekend
w_i^{34}	binary, whether disp. i works at least twice in period 23-05 for at least 2h/week **and** at least once on weekend

weekly rest of at least 36 hours starting during some of the periods. Constraint (8) enforces that there is no weekly rest period starting during day k for dispatcher i if they are assigned a shift j during k followed by a shift j' during $k+1$ between which we have less than 36 hours. Constraint (9) ensures that there is no weekly rest period starting during day k for dispatcher i, if they are assigned a shift j during k followed by a shift j' during $k+2$ between which we have less than 36 hours. Constraint (10) enforces that dispatcher i can be assigned at most one of two things during day $k+1$: a weekly rest period of 36 hours starting during day k and ending during day $k+2$, or a shift during day $k+1$.

Constraint (11) ensures that if dispatcher i is working some weekend shift, that is, a shift starting Saturday or Sunday (periods 6 and 7) or a shift starting on a Friday but ending on the Saturday, the variable indicating weekend work is set to 1. Constraint (12) enforces that if dispatcher i is not working any weekend shift, the variable indicating weekend work is set to 0. If a dispatcher is working either during a weekend or/and a shift during the week that overlaps with the time interval 20-06, the variable indicating this, w_i^{38}, is set to 1 with Constraint (13). If a dispatcher is working none of these shifts, the variable is set to 0 with Constraint (14). Similarly, Constraints (15) and (16) correctly set the variable indicating whether dispatcher i works at least twice during the hours 20-06; Constraints (17) and (18) correctly set the variable indicating whether dispatcher i works at least twice during the hours 20-06 and a weekend shift. Constraint (19) sums up the length of all shifts during the week assigned to dispatcher i and limits those plus 2 hours – if the dispatcher may by the other constraints work at most 38 hours – or plus 4 hours – if the dispatcher may work at most 36 hours – by the maximum weekly working hours of 40.

Constraints (20) and (21) keep track whether dispatcher worked at least two hours during 23 and 05 with the variable $\theta_i^{'23/05}$. Constraint (22) and (23) enforce that the variable indicating whether dispatcher i is working at least two hours within the hours 23-05 and at least once during the weekend, $w_i^{'36}$, is assigned correctly. If that variable is set to 1, Constraint (24) limits the lengths of shifts assigned to i to 36 (otherwise, the limit is 40 hours). Constraints (25) and (26) keep track whether dispatcher i worked at least twice during 23-05 with the variable $\theta_i^{23/05}$. Constraint (27) and (28) enforce that the variable indicating whether dispatcher i is working at least twice and for at least two hours within the hours 23-05 and at least once during the weekend, w_i^{34}, is assigned correctly. If that variable is set to 1, Constraint (29) limits the lengths of shifts assigned to i to 34[1].

4 Experiment Setup and Results

In this section, we present our experiments and their results followed by a discussion. We consider real-world-sized data, comparable to Malmö dispatching center, which is responsible for train traffic in southern Sweden. The dispatching area is partitioned into 15 dispatching areas (the area adjacency graph is given in Figure 2 in Appendix B) and the number of train movements per hour (approximating the dispatcher taskload) is based on discussions with operational experts, where we allow a maximum taskload of 30 per hour and dispatcher (see [16] for a detailed description of the train movements and the area's adjacencies). We assume that a dispatcher can handle a maximum of three areas simultaneously. We use one of our IPs for daily shifts [16, 15, 14] to create feasible sets of daily shifts. In Figure 3 in Appendix C, we present an example of the output of one of these models.

[1] More precisely, 34h and 15 minutes rounded down to 34h given the time resolution of 1h.

The input to the weekly problem consists of a set of shifts, S, represented by the start time, $start_j$, and end time, end_j, of each shift j. Using this data we compute the parameters for the weekly-scheduling model, such as ℓ_j, $\Delta_{j,j'}$, $\Delta'_{j,j'}$, $\delta_{j,j'}$, $\delta'_{j,j'}$, $o_j^{20/06}$, $o_j^{23/05}$, and $o'^{23/05}_j$. For all instances, we set the minimum resting time between two consecutive shifts, $r^{\min}$, to 11 periods (which equals 11 hours here), and the computational time limit to 48 hours.

We run three experiment series, varying the shift lengths and the number of night shifts (i.e., shifts overlapping with the period 20-06) in the daily-shifts input. In the first series, we change the interval of feasible shift lengths and the number of daily shifts when generating data for the daily-shifts model. In the second series, we change the distribution of the shift lengths, i.e., for each instance we generate daily shifts grouped by their length within predefined intervals. The purpose is to better control the structure of the shift lengths (short, medium, long) and to study the effect on the weekly dispatcher-shift assignments. Lastly, in the third series, we keep all parameters from the previous two series, in particular, the total number of daily shifts, while pushing for as few night shifts as possible (as night shifts strongly constrain the allowed weekly working hours). The purpose here is to examine the impact of the night-shift ratio on the weekly schedule.

For modeling and solving the IPs, we use the programming language Python and the solver Gurobi. As hardware, we use a powerful server (Tetralith server, 2019), utilizing Intel HNS2600BPB computer nodes with 32 CPU cores, 384 GB, provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS).

4.1 Series 1: Changing Shift Lengths and Total Number of Shifts

This first experiment series consists of two subseries (1.1 and 1.2). In the first one, we use a set of shifts with lengths within the interval [6-11], and we gradually increase the number of daily shifts. To generate these, we run the daily-shifts model where we set the lower ($T^{\min}$) and upper bound ($T^{\max}$) for the feasible shift length to 6 and 11 periods (which equal hours here), respectively. We start with 21 daily shifts, which is the minimum for a one-day coverage for that specific instance, and gradually increase this number to 25. The rationale for using increasing number of daily shifts is to create more puzzle bits and see how this will affect the quality of the weekly schedule. We use the notation $I_{t,n}^{[\ell b,ub]}$ for the instance with feasible shift lengths within the interval $[\ell b, ub]$, and where t is the total number of daily shifts and n is the number of night shifts among those. In the second experiment subseries (1.2), we adjust the feasible interval of the shift lengths in the daily-shifts model to [7-9]. The idea here is to avoid shifts that are either too short or too long and examine the impact of this change. The minimum number of shifts to cover one day of operations, for this specific instance, is 28. Similarly to Subseries 1.1, we gradually increase the number of daily shifts from 28 to 31.

We define the *slack* as the working hours of a dispatcher that we are not using with the current schedule. For example, if a dispatcher i has been assigned a shift pattern with a total of 34 hours, while according to the legal constraints for this specific shift pattern dispatcher i could have worked up to 38 hours, then the slack is 4 hours. We are not steering for a small slack with our model, but we use the slack as a performance indicator to gauge a shift pattern – with the general expectation that a schedule with large slack uses many dispatchers.

We report the shift-length distribution and the results of the two subseries in rows 2-10 of Table 3. We present the sum of slacks, their average per dispatcher, the total number of needed dispatchers, the optimality gap, and the runtime (rt) for each instance.

For both subseries, increasing the number of daily shifts in the input did not always result in a decrease in the number of needed dispatchers for the week. However, the number of dispatchers decreased between the instances $I_{21,16}^{[6,11]}$ and $I_{22,14}^{[6,11]}$. The first instance uses the minimum number of daily shifts, this makes it more likely that many shifts are pushed to the maximum of 11 hours. However, “puzzle bits” of 11 hours do not match well with the upper bounds on working hours of 34, 36, 38, and 40 hours. Thus, instances with a high number of 11-hour shifts are likely to yield large slacks. Increasing the number of daily shifts resulted in a decrease in the number of dispatchers in both $I_{22,14}^{[6,11]}$ and $I_{24,17}^{[6,11]}$, while this was not the case for $I_{23,18}^{[6,11]}$, and even worse for $I_{25,21}^{[6,11]}$, where the number of dispatchers increased to 49. These high numbers of dispatchers in these two instances are probably due to a high number of 11-hour shifts (15 in both cases). Increases/decreases in average slack are accompanied with increases/decreases in the number of 11-hour shifts.

The changes in daily shifts in the second subseries (1.2) yield smaller changes in the number of dispatchers, where the highest value (47) was obtained in $I_{29,22}^{[7,9]}$, which is the instance with the highest number of 9h-shifts in this subseries. The same instance has also the highest slack (both sum and average) within its group. Generally, the slack does not always follow the same trend as the number of dispatchers, which is the value we are minimizing.

From this experiment series, we conclude that adding more daily shifts may improve the weekly schedule, mainly because of adding shorter shifts at expense of longer ones. Too many 11-hour shifts may create a large number of weekly schedules with ca. 33 hours, which in turn yield a high slack. Hence, these experiments highlight the importance of the length-distribution of daily shifts.

The runtime reached the maximum limit of 48h in six out of nine instances with an optimality gap below 4.26. However, most instances reached the current solution relatively quickly, and the latest current solution has been reached before 9 hours.

Table 3 Length distribution and results for the first experiment series (1.1 and 1.2) and second experiment series.

	night	**slack**	**slack**	**slack**	**disp**	**gap**	***rt***	**length distr.**
instance name	**ratio**	$\sum$	**avg**	**min;max**	$\sum$	**%**	**(h)**	**6/7/8/9/10/11**
$I_{21,16}^{[6,11]}$	0.76	132	2.87	0;7	46	0	0.36	0/0/2/1/1/17
$I_{22,14}^{[6,11]}$	0.64	24	0.58	0;5	41	3.37	max	0/5/1/3/1/12
$I_{23,18}^{[6,11]}$	0.78	61	1.33	0;5	46	3.98	max	0/5/0/1/2/15
$I_{24,17}^{[6,11]}$	0.71	14	0.33	0;4	42	0	9.47	4/3/2/0/6/9
$I_{25,21}^{[6,11]}$	0.84	55	1.12	0;5	49	3.12	max	0/4/2/3/1/15
$I_{28,20}^{[7,9]}$	0.71	30	0.69	0;3	44	2.27	max	0/10/6/12/0/0
$I_{29,22}^{[7,9]}$	0.76	39	0.83	0;5	47	4.26	max	0/6/10/13/0/0
$I_{30,23}^{[7,9]}$	0.77	14	0.30	0;3	46	2.17	max	0/13/6/11/0/0
$I_{31,24}^{[7,9]}$	0.77	7	0.15	0;4	46	0	3.45	0/13/11/7/0/0
$I_{21,16}^{\frac{1}{2}:([6,8],[8,11])}$	0.76	7	0.19	0;1	37	0	0.73	0/1/10/1/0/9
$I_{22,17}^{\frac{1}{2}:([6,8],[8,11])}$	0.77	16	0.4	0;1	40	0	1.49	0/1/10/0/0/11
$I_{23,15}^{\frac{1}{2}:([6,8],[8,11])}$	0.65	12	0.3	0;1	40	0	1.50	0/4/8/0/1/10
$I_{21,16}^{\frac{1}{3}:([6,7],[7,9],[9,11])}$	0.76	3	0.08	0;1	36	0	3.19	0/7/0/7/0/7
$I_{22,15}^{\frac{1}{3}:([6,7],[7,9],[9,11])}$	0.68	6	0.16	0;1	37	0	1.48	0/8/0/7/0/7
$I_{23,16}^{\frac{1}{3}:([6,7],[7,9],[9,11])}$	0.69	12	0.31	0;2	39	0	5.80	1/6/1/8/1/6

4.2 Series 2: Splitting the Shift-Length Interval in Equal Subintervals

To generate more variability in the daily shifts, we adjust the input for the daily-shifts model, splitting the complete interval of allowed shift lengths into subintervals, and requiring roughly equally many shifts that have a shift length in each subinterval. This gave us the input for the second experiment series, containing the Subseries 2.1 and 2.2, which we report together with the corresponding results in rows 11-16 of Table 3. The instances of the first subseries series are $I_{t,n}^{(\frac{1}{2}:[6,8],[8,11])}$, where half[2] of the shifts have a length in the interval [6-8], and the other half have a length within [8-11]. We generated these shifts by adding extra constraints in the daily-shifts model. The two intervals overlap, because non-overlapping intervals may not yield feasible daily shifts when we keep all other parameters unchanged. The instances in the second subseries are $I_{t,n}^{(\frac{1}{3}:[6,7],[7,9],[9,11])}$, where we split the shifts into three equinumerous sets with lengths within the intervals [6,7], [7,9] and [9,11], respectively (again we allow overlapping intervals)[3]. In both subseries, we have instances with 21, 22, and 23 daily shifts.

Both the slack and the number of dispatchers improve in each instance $I_{t,n}^{\frac{1}{2}:([6,8],[8,11])}$ compared to its correspondent in Subseries 1.1 with the same number of daily shifts, $I_{t,n}^{[6,11]}$. Comparing the instances only within this subseries, $I_{21,16}^{\frac{1}{2}:([6,8],[8,11])}$ performs best, which is probably at least partly due to relatively few long (11-hour) shifts. The same trend holds for instances $I_{t,n}^{\frac{1}{3}:([6,7],[7,9],[9,11])}$ compared to their correspondent instances $I_{t,n}^{[6,11]}$. Every instance in the second experiment series outperformed any one in the first series w.r.t. the number of used dispatchers. Moreover, the runtimes are generally much shorter in this second series.

4.3 Series 3: Changing the Percentage of Night Shifts

In all the previous experiments, relatively few shifts covered only daytime, while the majority overlapped with night hours. In this experiment series, we aim to investigate the impact of changing the ratio of the number of night shifts and the total number of shifts by decreasing the number of night shifts (while keeping the total number of daily shifts constant). For each of the previous instances, we generate a corresponding instance with the lowest night-shift ratio for the given total number of daily shifts. The idea here is to minimize the number of shifts involved in the constraints on the total weekly working hours, hence, we define the night shifts as any shift that overlaps with the period 20-06 (we cannot impact the weekend shifts, as all daily shifts have to be manned during the weekend). To generate these shifts in the daily-shifts model, we prohibit some shifts from covering the time interval 20-06, performing a binary search for the highest number of daily shifts that are prohibited to overlap with the night hours, while yielding a feasible solution. In Table 4, we report the shift distribution and the results of these instances, which we denote by I' instead of I.

The number of dispatchers in instance I' is lower than in instance I in 12 out of 15 instances; this number remains unchanged in two cases; and only for $I'^{[6,11]}_{22,12}$, the number is higher than in $I^{[6,11]}_{22,12}$. This sole increase in the number of dispatchers is probably caused by an increase in the number of 11-hour shifts.

The slack decreased for all but three instances I' in comparison to the corresponding instance I (these three instances are marked in red in Table 4). However, in two out of three cases, the increase in slack is accompanied by a decrease in the number of dispatchers (our objective). Hence, the trend in slack is often a good indicator for the trend in the number of needed dispatchers, but this does not always hold.

[2] More precisely, we have $\lceil \frac{t}{2} \rceil$ daily shifts in one shift-length interval and $\lfloor \frac{t}{2} \rfloor$ in the other interval.

[3] And again, we actually have two shift-length intervals with $\lfloor \frac{t}{3} \rfloor$ shifts and one shift-length interval with $\lceil \frac{t}{3} \rceil$ shifts.

Table 4 Results for the third experiment series (with adjusted parameters of all instances in the first two series), where the number of night shifts is minimized. For instances I' with a higher slack than the corresponding instance I, we highlight the slack in red; for instances with a higher number of dispatchers, we highlight the number in bold.

	night	**slack**	**slack**	**slack**	**disp**	**gap**	*rt*	**length distr.**
instance name	**ratio**	$\sum$	**avg**	**min;max**	$\sum$	%	(h)	**6/7/8/9/10/11**
$I'^{[6,11]}_{21,13}$	0.62	30	0.77	0;5	39	2.56	max	1/0/1/1/4/14
$I'^{[6,11]}_{22,12}$	0.54	46	1.07	0;5	**43**	2.33	max	2/1/0/2/2/15
$I'^{[6,11]}_{23,11}$	0.48	4	0.1	0;2	38	0	1.42	5/5/0/2/2/9
$I'^{[6,11]}_{24,10}$	0.42	10	0.24	0;4	41	0	2.13	4/4/1/3/1/11
$I'^{[6,11]}_{25,9}$	0.36	1	0.02	0;1	40	0	3.52	7/4/2/1/1/10
$I'^{[7,9]}_{28,13}$	0.46	43	1.02	0;8	42	0	30.28	0/16/5/7/0/0
$I'^{[7,9]}_{29,13}$	0.49	20	0.44	0;5	45	2.22	max	0/10/5/14/0/0
$I'^{[7,9]}_{30,13}$	0.43	12	0.26	0;3	46	2.17	max	0/9/10/11/0/0
$I'^{[7,9]}_{31,13}$	0.42	2	0.04	0;1	46	2.17	max	0/13/9/9/0/0
$I'^{\frac{1}{2}:([6,8],[8,11])}_{21,13}$	0.62	8	0.22	0;2	36	2.78	max	2/3/6/0/1/9
$I'^{\frac{1}{2}:([6,8],[8,11])}_{22,12}$	0.54	4	0.11	0;1	37	0	1.59	6/1/4/0/2/9
$I'^{\frac{1}{2}:([6,8],[8,11])}_{23,12}$	0.52	9	0.23	0;3	39	0	0.18	6/1/5/0/1/10
$I'^{\frac{1}{3}:([6,7],[7,9],[9,11])}_{21,13}$	0.62	1	0.03	0;1	35	0	2.94	1/6/1/6/0/7
$I'^{\frac{1}{3}:([6,7],[7,9],[9,11])}_{22,12}$	0.54	3	0.08	0;1	36	0	0.28	4/4/1/6/0/7
$I'^{\frac{1}{3}:([6,7],[7,9],[9,11])}_{23,12}$	0.52	5	0.13	0;1	38	0	2.61	2/6/1/7/1/6

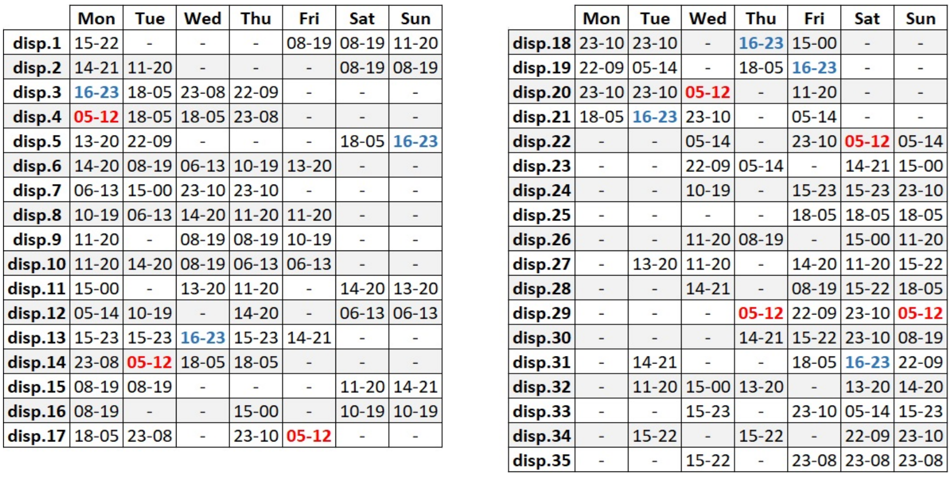
Since we are not steering the shift-length distribution in this experiment series, we exclude the instances for which not only the number of night shifts changed, but for which the number of the longest shifts decreased. The rationale behind this is that changes in both would not allow us to determine whether the improvements are due to having fewer night shifts. Considering only instances where the number of longest shifts did not decrease (10 instances), only two out of these ten instances, namely $I'^{[6,11]}_{22,12}$ and$I'^{\frac{1}{2}:([6,8],[8,11])}_{21,13}$, had worse results either in terms of increased slack or both increased slack and number of dispatchers. The remaining eight out of ten instances yield better results, which is possibly because of fewer night shifts.

In Figure 1, we present the final weekly schedule obtained for instance $I'^{\frac{1}{3}:([6,7],[7,9],[9,11])}_{21,16}$. This is the instance with the lowest number of needed dispatchers (35).

Generally, minimizing the number of night shifts has a positive effect on the weekly schedule, which is in line with our expectations, since having more night shifts would activate more constraints that limit the weekly working hours, thus, increasing the number of needed dispatchers.

5 Conclusions and Future Work

Solving shift-scheduling problems with a long time horizon and a high resolution is a complex problem that requires a huge amount of computation time, and in many cases depending on the size and the problem type, without reaching a global optimum. In this paper, we suggest a two-stage approach for weekly shift scheduling of train dispatchers. First, we use our previous model from [16, 15, 14] to generate feasible daily shifts, which we then use as input to the weekly-shift-scheduling model presented in this paper. We run three experiment



	Mon	Tue	Wed	Thu	Fri	Sat	Sun
disp.1	15-22	-	-	-	08-19	08-19	11-20
disp.2	14-21	11-20	-	-	-	08-19	08-19
disp.3	**16-23**	18-05	23-08	22-09	-	-	-
disp.4	**05-12**	18-05	18-05	23-08	-	-	-
disp.5	13-20	22-09	-	-	-	18-05	**16-23**
disp.6	14-20	08-19	06-13	10-19	13-20	-	-
disp.7	06-13	15-00	23-10	23-10	-	-	-
disp.8	10-19	06-13	14-20	11-20	11-20	-	-
disp.9	11-20	-	08-19	08-19	10-19	-	-
disp.10	11-20	14-20	08-19	06-13	06-13	-	-
disp.11	15-00	-	13-20	11-20	-	14-20	13-20
disp.12	05-14	10-19	-	14-20	-	06-13	06-13
disp.13	15-23	15-23	**16-23**	15-23	14-21	-	-
disp.14	23-08	**05-12**	18-05	18-05	-	-	-
disp.15	08-19	08-19	-	-	-	11-20	14-21
disp.16	08-19	-	-	15-00	-	10-19	10-19
disp.17	18-05	23-08	-	23-10	**05-12**	-	-

	Mon	Tue	Wed	Thu	Fri	Sat	Sun
disp.18	23-10	23-10	-	**16-23**	15-00	-	-
disp.19	22-09	05-14	-	18-05	**16-23**	-	-
disp.20	23-10	23-10	**05-12**	-	11-20	-	-
disp.21	18-05	**16-23**	23-10	-	05-14	-	-
disp.22	-	-	05-14	-	23-10	**05-12**	05-14
disp.23	-	-	22-09	05-14	-	14-21	15-00
disp.24	-	-	10-19	-	15-23	15-23	23-10
disp.25	-	-	-	-	18-05	18-05	18-05
disp.26	-	-	11-20	08-19	-	15-00	11-20
disp.27	-	13-20	11-20	-	14-20	11-20	15-22
disp.28	-	-	14-21	-	08-19	15-22	18-05
disp.29	-	-	-	**05-12**	22-09	23-10	**05-12**
disp.30	-	-	-	14-21	15-22	23-10	08-19
disp.31	-	14-21	-	-	18-05	**16-23**	22-09
disp.32	-	11-20	15-00	13-20	-	13-20	14-20
disp.33	-	-	15-23	-	23-10	05-14	15-23
disp.34	-	15-22	-	15-22	-	22-09	23-10
disp.35	-	-	15-22	-	23-08	23-08	23-08

Figure 1 The weekly schedule obtained for instance $I'^{\frac{1}{3}:([6,7],[7,9],[9,11])}_{21,16}$. The values in the cells represent the start and end time of each shift. To exemplify the daily assignment of all daily shifts, we marked two of the 21 daily shifts, the shifts 05-12 and 16-23 in red and blue, respectively.

series, with a real-world sized data, where we: change the number of daily shift and their feasible lengths, split the feasible shift-length interval into equal subintervals, and change the percentage of night shifts among the daily shifts.

We conclude that increasing the number of daily shifts may improve the weekly schedules, especially if this would decrease the percentage of the too long shifts (11h). Moreover, enforcing more variability in the daily shifts' length, and reducing the night-shift ratio can also improve the quality of the weekly schedules.

For stage two in our approach, the runtimes in our experiments were between a few minutes up to being stopped after two days with an optimality gap below 4.26%. The instances with minimized night shift ratio and with three subintervals of the feasible lengths performed very well, and also had a relatively low runtime (between 0.28 and 2.94h). Thus, we recommend those daily-shift patterns. For stage one in our approach, computing one feasible set of daily shifts – using a model to minimize the dispatcher-area-assignment switches [15, 14] – takes less than two hours. However, if we want to achieve the fewest possible night shifts, we may have to perform multiple runs. Minimizing the number of dispatchers only is much faster (with a runtime of maximum 20 seconds). We could use this objective for our daily-shifts model to obtain the minimum number of night shifts (performing a binary search on that number with a few seconds runtime per run) and then once run the computationally more expensive model to minimize dispatcher-area-assignment switches; with this, we would prioritize the smallest number of night shifts over the lowest possible number of assignment switches. Given the planning horizon, the resulting runtime for the complete approach can be acceptable for real-world use, in particular, given the quality of the resulting schedules.

We opted for a two-stage approach, mainly to gain insight not only in the weekly schedules, but also in "good" daily-shift distributions for the operational planning. Another natural approach is column generation, where both constraints on the daily shifts and those on the weekly shifts are integrated. The split in master and pricing problem could, for example, be steered by operational vs. legal constraints on the shifts, respectively.

References

1 Burak Bilgin, Patrick De Causmaecker, Benoît Rossie, and Greet Vanden Berghe. Local search neighbourhoods for dealing with a novel nurse rostering model. *Annals of Operations Research*, 194:33–57, 2012.

2 Michael J Brusco and TR Johns. An integrated approach to shift-starting time selection and tour-schedule construction. *Journal of the Operational Research Society*, 62(7):1357–1364, 2011.

3 Edmund K Burke, Timothy Curtois, Gerhard Post, Rong Qu, and Bart Veltman. A hybrid heuristic ordering and variable neighbourhood search for the nurse rostering problem. *European journal of operational research*, 188(2):330–341, 2008.

4 Mehmet Tolga Cezik and Pierre L'Ecuyer. Staffing multiskill call centers via linear programming and simulation. *Management Science*, 54(2):310–323, 2008.

5 Jean-François Cordeau, Gilbert Laporte, Federico Pasin, and Stefan Ropke. Scheduling technicians and tasks in a telecommunications company. *Journal of Scheduling*, 13(4):393–409, 2010.

6 Sophie Demassey, Gilles Pesant, and Louis-Martin Rousseau. Constraint programming based column generation for employee timetabling. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems: Second International Conference, CPAIOR 2005, Prague, Czech Republic, May 31-June 1, 2005. Proceedings 2*, pages 140–154. Springer, 2005.

7 Sophie Demassey, Gilles Pesant, and Louis-Martin Rousseau. A cost-regular based hybrid column generation approach. *Constraints*, 11(4):315–333, 2006. `doi:10.1007/s10601-006-9003-7`.

8 Robert G Drake. The nurse rostering problem: from operational research to organizational reality? *Journal of advanced nursing*, 70(4):800–810, 2014.

9 eurostat. Passenger transport by type of transport (detailed reporting only), 2024. URL: `https://ec.europa.eu/eurostat/databrowser/view/RAIL_PA_TYPEPAS/default/line?lang=en`.

10 Michel Gamache, François Soumis, Gérald Marquis, and Jacques Desrosiers. A column generation approach for large-scale aircrew rostering problems. *Operations Research*, 47(2):247–263, 1999. `doi:10.1287/opre.47.2.247`.

11 Jia Guo and Jonathan F Bard. Air traffic controller scheduling. *Computers & Industrial Engineering*, 191:110123, 2024.

12 Dennis Huisman. A column generation approach for the rail crew re-scheduling problem. *European Journal of Operational Research*, 180(1):163–173, 2007. `doi:10.1016/j.ejor.2006.04.026`.

13 Gilbert Laporte and Gilles Pesant. A general multi-shift scheduling system. *Journal of the Operational Research Society*, 55(11):1208–1217, 2004.

14 Tomas Lidén, Christiane Schmidt, and Rabii Zahir. Improving attractiveness of working shifts for train dispatchers. Under revision for journal publication.

15 Tomas Lidén, Christiane Schmidt, and Rabii Zahir. Improving attractiveness of working shifts for train dispatchers. In *25th Euro Working Group on Transportation Meeting (EWGT 2023)*, 2023. Extended abstract presented only.

16 Tomas Lidén, Christiane Schmidt, and Rabii Zahir. Shift scheduling for train dispatchers. In *RailBelgrade 2023: the 10th International Conference on Railway Operations Modelling and Analysis (ICROMA)*, 2023.

17 Nysret Musliu, Andrea Schaerf, and Wolfgang Slany. Local search for shift design. *European journal of operational research*, 153(1):51–64, 2004.

18 Emilie M. Roth, Nicholas Malsch, and J. Multer. Understanding how train dispatchers manage and control trains : results of a cognitive task analysis. Technical Report DOT-VNTSC-FRA-98-3;DOT/FRA/ORD-01/02, John A. Volpe National Transportation Systems Center (U.S.), 2001.

19 Pieter Smet, Burak Bilgin, Patrick De Causmaecker, and Greet Vanden Berghe. Modelling and evaluation issues in nurse rostering. *Annals of Operations Research*, 218:303–326, 2014.

20 Trafikverket. Kollektivavtal: Trafikverkets affärsverksavtal mellan Trafikverket och Saco-S, OFR-S och Seko, 2019. URL: `https://www.seko.se/siteassets/forhandlings-branschorganisationer/seko-klubb-trafikverket/pdf-er/test/trafikverkets-affarsverksavtal_andringar_tillagg_till_och_med_2019_10_01_20191025.pdf`.

21 Trafikverket. Sveriges järnvägsnät, 2024. Accessed: 2024-06-26. URL: `https:https://www.trafikverket.se/resa-och-trafik/jarnvag/sveriges-jarnvagsnat/`.

A Mathematical Model for Daily Shifts

For the ease of the reader, we provide the full model used for computing the daily shifts that we presented in [15, 14] (an improvement of the model from [16]) in this section of the appendix. While we also presented different approaches for minimizing changes in the dispatcher-area assignment in [15, 14], here, we give the improved model for minimizing the number of used dispatchers while observing all operational and legal requirements for shifts during one day.

We give the notation we used in the model in Tables 5 and 6. Because we aim to minimize the total number of used dispatchers, our objective function is given by (31).

$$\min. \quad \sum_{i \in D} q_i \tag{31}$$

The rest of the model is given by Constraints (32)-(43).

Table 5 Model parameters in the one-day-shifts model.

Parameters	Description
D	set of train dispatchers, indexed by i
A	set of geographical areas, indexed by j
P	set of time periods, indexed by k
C	set of area combinations, indexed by ℓ
$TL_{j,k}$	task load in area j during period k
$TL^{\max}$	maximum allowed task load
$A^{\max}$	maximum number of assigned areas to a dispatcher per period
$e_{i,j} \in \{0,1\}$	=1 if dispatcher i holds an endorsement for area j
$T^{\min}$	minimum shift length (in time periods)
$T^{\max}$	maximum shift length (in time periods)
$r^{\min}$	minimum number of rest periods between two shifts
$p = \lvert P\rvert$	number of time periods in the time horizon

Table 6 Model variables in the one-day-shifts model.

Variables	Description
$x_{i,j,k} \in \{0,1\}$	=1 if dispatcher i is assigned area j during period k
$c_{i,\ell,k} \in \{0,1\}$	=1 if dispatcher i is assigned area combination ℓ during period k
$y_{i,k} \in \{0,1\}$	=1 if dispatcher i is at work during period k
$v_{i,k} \in \{0,1\}$	=1 if dispatcher i starts a shift at the beginning of period k
$q_i \in \{0,1\}$	=1 if dispatcher i is used during some period

$$\sum_{\mu=k+1-T^{\min}}^{k} v_{i,\mu \pmod p} \leq y_{i,k} \qquad \forall i \in D, \forall k \in P \tag{32}$$

$$\sum_{\mu=k+1-T^{\max}}^{k} v_{i,\mu \pmod p} \geq y_{i,k} \qquad \forall i \in D, \forall k \in P \tag{33}$$

$$v_{i,k} \geq y_{i,k} - y_{i,(k-1) \pmod p} \qquad \forall i \in D, \forall k \in P \tag{34}$$

$$v_{i,k} \leq y_{i,k} \qquad \forall i \in D, \forall k \in P \tag{35}$$

$$v_{i,k} \leq q_i \qquad \forall i \in D, \forall k \in P \tag{36}$$

$$\sum_{k \in P} v_{i,k} \geq q_i \qquad \forall i \in D \tag{37}$$

$$\sum_{\mu=k+1}^{k+r^{\min}} v_{i,\mu \pmod p} \leq q_i - y_{i,k} \qquad \forall i \in D, \forall k \in P \tag{38}$$

$$\sum_{j \in A} \sum_{\ell \in C} c_{i,\ell,k} \cdot a_{\ell,j} \cdot TL_{j,k} \leq TL^{\max} \qquad \forall i \in D, \forall k \in P \tag{39}$$

$$c_{i,\ell,k} \leq e_{i,j} \qquad \forall i \in D, \forall \ell \in C, \forall j \in \ell, \forall k \in P \tag{40}$$

$$\sum_{\ell \in C \setminus \{0\}} c_{i,\ell,k} = y_{i,k} \qquad \forall i \in D, \forall k \in P \tag{41}$$

$$c_{i,0,k} = 1 - y_{i,k} \qquad \forall i \in D, \forall k \in P \tag{42}$$

$$\sum_{\ell \in C \setminus \{0\}} \sum_{i \in D} a_{\ell,j} \cdot c_{i,\ell,k} = 1 \qquad \forall k \in P, \forall j \in A \tag{43}$$

B Adjacency Graph of the Considered Dispatching Areas

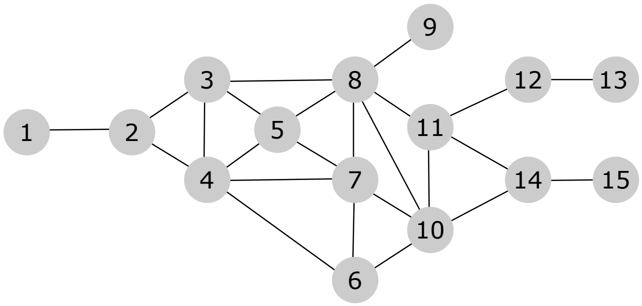
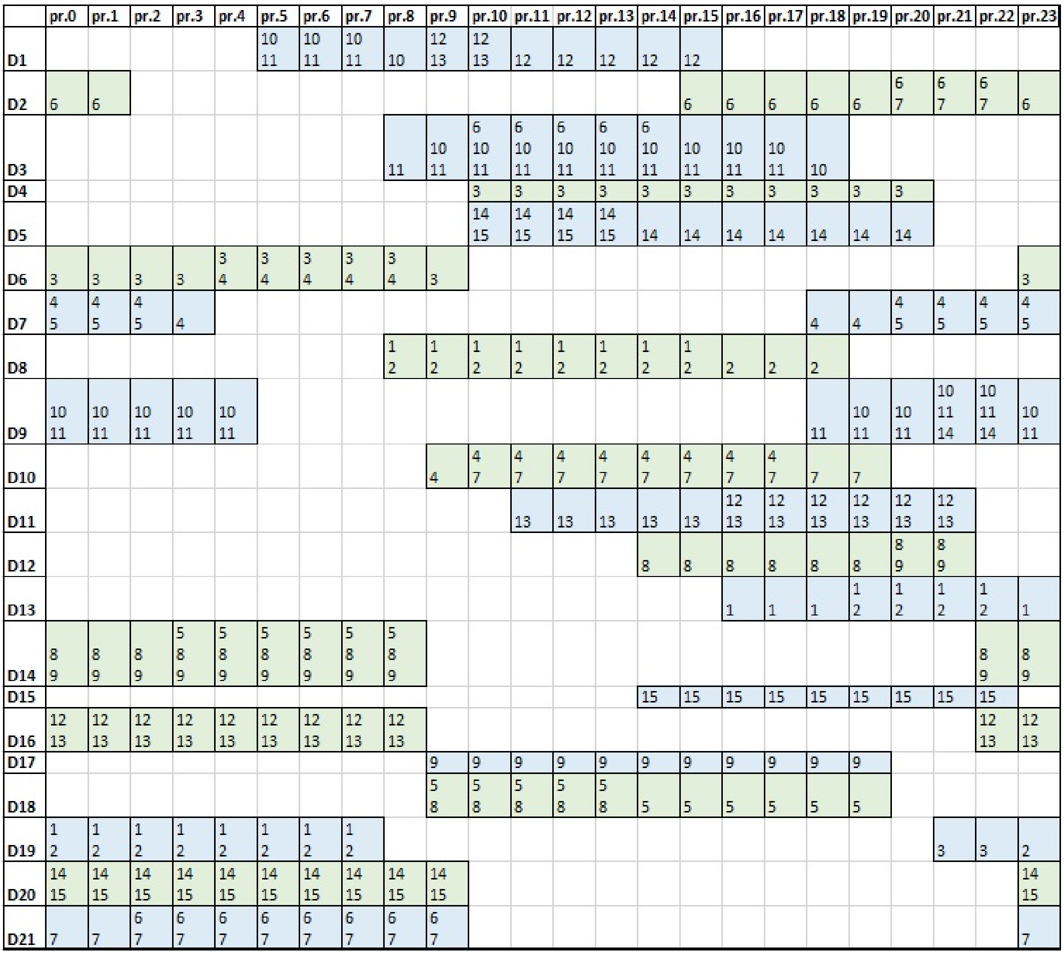


Figure 2 Adjacency graph of the considered dispatching areas.

C Example Output of the Daily-Shifts Model



	pr.0	pr.1	pr.2	pr.3	pr.4	pr.5	pr.6	pr.7	pr.8	pr.9	pr.10	pr.11	pr.12	pr.13	pr.14	pr.15	pr.16	pr.17	pr.18	pr.19	pr.20	pr.21	pr.22	pr.23
D1						10 11	10 11	10 11	10	12 13	12 13	12	12	12	12	12								
D2	6	6														6	6	6	6	6	6 7	6 7	6 7	6
D3									11	10 11	6 10 11	6 10 11	6 10 11	6 10 11	6 10 11	10 11	10 11	10 11	10					
D4											3	3	3	3	3	3	3	3	3	3	3			
D5											14 15	14 15	14 15	14 15	14	14	14	14	14	14	14			
D6	3	3	3	3	3 4	3 4	3 4	3 4	3 4	3														3
D7	4 5	4 5	4 5	4															4	4	4 5	4 5	4 5	4 5
D8									1 2	1 2	1 2	1 2	1 2	1 2	1 2	1 2	2	2	2					
D9	10 11	10 11	10 11	10 11	10 11														11	10 11	10 11	10 11 14	10 11 14	10 11
D10										4	4 7	4 7	4 7	4 7	4 7	4 7	4 7	4 7	7	7				
D11												13	13	13	13	13	12 13	12 13	12 13	12 13	12 13	12 13		
D12															8	8	8	8	8	8	8 9	8 9		
D13																	1	1	1	1 2	1 2	1 2	1 2	1
D14	8 9	8 9	8 9	5 8 9	5 8 9	5 8 9	5 8 9	5 8 9	5 8 9														8 9	8 9
D15															15	15	15	15	15	15	15	15	15	
D16	12 13	12 13	12 13	12 13	12 13	12 13	12 13	12 13	12 13														12 13	12 13
D17										9	9	9	9	9	9	9	9	9	9	9				
D18										5 8	5 8	5 8	5 8	5 8	5	5	5	5	5	5				
D19	1 2	1 2	1 2	1 2	1 2	1 2	1 2	1 2														3	3	2
D20	14 15	14 15	14 15	14 15	14 15	14 15	14 15	14 15	14 15	14 15														14 15
D21	7	7	6 7	6 7	6 7	6 7	6 7	6 7	6 7	6 7														7

Figure 3 An optimum schedule obtained by the daily-shifts model given an instance with 24 periods and 15 areas. Rows and columns stand for dispatchers and time period of a day, respectively. The values in the cells represent the assigned areas, while the colors are used for clarity and distinction between rows.

Improved Algorithms for the Capacitated Team Orienteering Problem

Gianlorenzo D'Angelo ✉

Gran Sasso Science Institute, L'Aquila, Italy

Mattia D'Emidio ✉

Dept. of Information Engineering, Computer Science and Mathematics,
University of L'Aquila, L'Aquila, Italy

Esmaeil Delfaraz ✉

Dept. of Information Engineering, Computer Science and Mathematics,
University of L'Aquila, L'Aquila, Italy

Gabriele Di Stefano ✉

Dept. of Information Engineering, Computer Science and Mathematics,
University of L'Aquila, L'Aquila, Italy

Abstract

We study the Capacitated Team Orienteering Problem, where a fleet of vehicles with capacities have to meet customers with known demands and prizes for a single commodity. The objective is to maximize the total prize and to assign a sequence of customers to each vehicle while keeping the total distance traveled within a given budget and such that the total demand served by each vehicle does not exceed its capacity. The problem has been widely studied both from a theoretical and a practical point of view. The contribution of this paper is twofold: (1) We advance the theoretical knowledge on the problem by providing new approximation algorithms that achieve, under some natural assumption, improved approximation ratios compared to the current best algorithms; (2) We propose four efficient heuristics that outperform the current state-of-the-art practical methods in the sense that they compute solutions that collect nearly the same prize in a significantly smaller running time. We also experimentally test the scalability of the new heuristics, showing that their running time increases approximately linearly with the size of the input, allowing us to process large graphs which were not possible to analyze before.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Theory of computation → Approximation algorithms analysis; Theory of computation → Design and analysis of algorithms

Keywords and phrases Vehicle Routing, Approximation algorithms, Algorithm Engineering

Digital Object Identifier 10.4230/OASIcs.ATMOS.2024.7

Funding This work was partially supported by: Project EMERGE: innovation agreement between Ministry of Economical Development, Abruzzo Region, Radiolabs, Elital, Leonardo, Telespazio, University of L'Aquila and Centre of EXcellence EX-Emerge, funded by Italian Government under CIPE n. 70/2017; Group for Scientific Computation of Istituto Nazionale di Alta Matematica (GNCS-INdAM); the European Union - NextGenerationEU under the Italian Ministry of University and Research National Innovation Ecosystem grant ECS00000041 - VITALITY - CUP: D13C21000430001; PNRR MUR project GAMING "Graph Algorithms and MinINg for Green agents" (PE0000013, CUP D13C24000430001); the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on "Telecommunications of the Future" (PE00000001 - program RESTART), project MoVeOver/SCHEDULE ("Smart interseCtions witH connEcteD and aUtonomous vehicLEs", CUP J33C22002880001); ARS01_00540 - RASTA project, funded by the Italian Ministry of Research PNR 2015–2020. The first author acknowledges the support of the MUR (Italy) Department of Excellence 2023–2027.

24th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2024).
Editors: Paul C. Bouman and Spyros C. Kontogiannis; Article No. 7; pp. 7:1–7:17
OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The *Capacitated Orienteering Problem* (C-OP) is an NP-hard combinatorial optimization problem belonging to the wide class of Vehicle Routing Problems (VRPs), which has received much attention in the literature on algorithms and operation research [2,6,8,13,16–19,22,25,26]. In C-OP, we are given a complete graph, with edge lengths, where each node represents a customer that is assigned a profit/prize and a demand/size. Given two nodes s and t the goal is to find a path from s to t that maximizes the total prize, and respects both a capacity constraint on the total size of the nodes on the s-t path and a budget constraint on the total length of the s-t path. C-OP is a natural generalization of two very well-known problems, namely the *Knapsack Problem* [27], which is a special case of C-OP where the length of all edges is zero, and the *Orienteering Problem* (OP) [9,10], which is a variant of C-OP where the size of all nodes is zero. A generalization of C-OP is the case in which the goal is to find s-t paths for a fleet of K homogeneous, capacitated vehicles that can be used to collect prizes. This problem is known as the *Capacitated Team Orienteering Problem* (C-TOP) and has been defined by Archetti et al. [2], originally in the flavor when $s = t$, i.e. when the aim is finding tours centered at a depot node, rather than paths.

From a theoretical viewpoint, the best known approximation algorithms for C-OP and C-TOP, that run in polynomial time, are due to Bock and Sanità, who achieved approximation factors of $(3 + \varepsilon)$, for any $\varepsilon > 0$, and 3.53, respectively [8]. From a practical perspective, for both C-OP and C-TOP, several heuristics without guarantees on the achieved quality of solution and exact algorithms with exponentially large worst-case running times have been introduced and experimentally evaluated with the aim of characterizing their practical effectiveness and applicability, i.e. evaluating the quality of the computed solutions and the running time necessary to achieve such solutions (see [1, 2, 6, 16, 17, 19, 22, 25, 26]). In all benchmark instances for the C-OP and C-TOP problems considered in such works, the prize of each node is fixed to be at least equal to half of its size. Specifically, the prize of each node v with size $r(v)$ is assigned to be equal to $\pi(v) = (h + 0.5)r(v)$, where h is a random number uniformly generated within interval $[0, 1]$ [2, 26]. This implies that, for two nodes u and v with $r(u) \geq r(v)$, we have $\pi(u) \geq \pi(v)/3$. Motivated by this observation, we consider problems C-OP and C-TOP under the natural assumption that choosing subsets of nodes with larger sizes results in achieving (almost) more prizes. In more detail, we assume that any subset S of nodes collects an overall prize that is at least equal to a multiplicative factor $\lambda \in (0, 1]$ times the prize collected by any subset of nodes whose sum of sizes is lower than the sum of the sizes of nodes in S (see Assumption 2.1 for a formal definition).

Our Contribution. The contribution of this paper is both theoretical and experimental. From the theoretical viewpoint, we improve over the state-of-the art by providing approximation algorithms, for C-OP and C-TOP, that guarantees an approximation ratio which, under particular assumptions, is smaller than the best approximation ratio known so far. In particular, we propose a $\max\{\alpha, \frac{2}{\lambda}\}$-approximation algorithm for C-OP and a $(1 - e^{-\frac{1}{\beta}})^{-1}$-approximation algorithm for C-TOP, where α is the approximation factor of an algorithm for OP, $\beta = \max\{\alpha, \frac{2}{\lambda}\}$, and λ is the parameter of Assumption 2.1. Observe that, the best known approximation algorithm for OP is that given by Chekuri et al. [9], which guarantees an approximation factor of $2 + \varepsilon$, for any $\varepsilon > 0$. When $\lambda \in [\frac{2}{3}, 1]$, our algorithms for C-OP and C-TOP achieve approximation factors in the intervals $[2 + \varepsilon, 3]$ and $[2.55, 3.53)$, respectively, for any $\varepsilon > 0$. These improve over the long-standing results by Bock and Sanità [8] who achieved factors $3 + \varepsilon$ and 3.53 for C-OP and C-TOP, respectively, for any $\varepsilon > 0$.

Since our algorithms with theoretical guarantees have high computational complexity, we propose four efficient heuristic algorithms that do not give any proven guarantee on the quality of the computed solution but achieve good performance in practice. We experimentally evaluate our heuristics in benchmark instances from the literature and show that two of them produce solutions that are comparable to the best solutions known from the literature in terms of collected prize, but outperform all the state-of-the art algorithms in terms of running time. In particular, our heuristics require less than a second on the small instances (at most 100 nodes) and two orders of magnitude less time than other algorithms on large instances (at most 577 nodes), while achieving the same prize in most cases, a slightly worse prize in a few cases, and even a better prize in a few cases. To assess how the time performance of our heuristics scales with the input size, we also generated new instances with up to 15 500 nodes, starting from real-world road networks. Our experiments in these instances suggest that the running time of two of our algorithms tends to grow approximately linearly with the input size and highlight that, on the largest instance, such two algorithms take below one minute on average, whereas previous algorithms are not able to handle such large input graphs.

Related Work. Blum et al. [7] gave the first constant factor approximation algorithm for OP with approximation ratio of 4 when $s = t$ and showed that: (i) no polynomial-time approximation algorithm can achieve a factor better than $\frac{1481}{1480}$; (ii) OP is APX-hard. Bansal et al. [5] improved the bound of [7] by designing a 3-approximation algorithm for the case where $s = t$ while Chekuri et al. [9] proposed a $(2 + \varepsilon)$-approximation algorithm that works for any positive constant ε. Friggstad and Swamy [15] designed, via LP-rounding, a 3-approximation algorithm when $s = t$. Paul et al. [23], gave a 2-approximation algorithms for OP when s and t are not given in advance. Finally, Chen and Har-Peled [10] gave a PTAS for the case where the points lie in a constant-dimensional Euclidean metric space.

A natural generalization of OP is the Team Orienteering Problem (TOP) where we are asked to find $K \geq 1$ paths from s to t that maximize the total prize, accumulated by all the K paths, and such that each path respects the budget B. Blum et al. [7] studied TOP under the name of *Multi-Path Orienteering problem* (M-OP) and showed that: (i) any α approximation for OP, when $s = t$, can be translated into a $1/(1 - e^{-\alpha})$ approximation for M-OP; (ii) their algorithm for M-OP has a factor of $\alpha + 1$ when the starting point of each vehicle is arbitrary. Friggstad et al. [14] studied a variant of M-OP in the case where each vehicle needs to find a tour and each node has a cost. The goal is to find K tours so that the minimum total prize among all tours is maximized, i.e. to find $P' : \pi(P') = \max \min_P \pi(P)$. They called this problem *max-min* orienteering and showed that any α-approximation algorithm for OP results in an $(\alpha + 2)$-approximation for *max-min* orienteering. Xu et al. [28] studied a variant of TOP in which the prize function is a special submodular function and showed the existence of a $1/(1 - e^{-\alpha})$-approximation algorithm for such variant, where α is an approximation factor to OP. Finally, Xu et al. [29] focused on TOP when $s = t$, they call this variant the *monitoring reward maximization* problem and presented a 3-approximation algorithm. Clearly, C-OP is a generalization of OP in which we also consider node demands $r : V \to \mathbb{N}$ and a capacity bound C. Gupta et al. [18] showed that, given an α-approximation algorithm for OP, it is possible to derive a 2α-approximation algorithm for C-OP. By using the $(2 + \varepsilon)$-approximation algorithm for OP [9], this leads to a $(4 + \varepsilon)$-approximation for C-OP. Bock and Sanità [8] improved this result by giving a $(1 + \alpha + \varepsilon)$-approximation algorithm for C-OP and by presenting a PTAS on trees and a PTAS on Euclidean metrics. Again, using the $(2 + \varepsilon)$-approximation algorithm for OP, results in a $(3 + \varepsilon)$-approximation for C-OP. For C-TOP, Bock and Sanità [8] designed a $(1 - e^{\frac{1}{\beta}})$-approximation algorithm, where β is an approximation factor for C-OP. Using $\beta = 3 + \varepsilon$ this leads to a 3.53-approximation algorithm for C-TOP.

2 Notation and Definitions

We are given an undirected complete graph $G = (V, E)$ with $n = |V|$ vertices and $m = |E|$ edges, respectively. Let $l : E \to \mathbb{R}_{\geq 0}$ be a metric *length* function on edges, let $\pi : V \to \mathbb{R}_{\geq 0}$ be a *prize* function on the nodes, let $r : V \to \mathbb{R}_{\geq 0}$ be a *size* function on the nodes, and let $g : V \to \mathbb{R}_{\geq 0}$ be a *service time* function on the nodes. For any subgraph G' of G, we denote by $V(G')$ and $E(G')$ the set of nodes and edges in G', respectively. Given a subset $S \subseteq V$, $G[S]$ denotes the subgraph of G induced by S, i.e., $E(G[S]) = \{\{u, v\} \in E \mid u, v \in S\}$.

For an integer k, let $[k] := \{1, 2, \ldots, k\}$. A *path* P_{uv} from node u to node v is a graph made of a sequence of distinct nodes $\{v_1 = u, \ldots, v_k = v\}$ and a sequence of edges $\{v_i, v_{i+1}\}$, where $i \in [k-1]$. The cost of a path P_{uv} in G is the sum of the lengths of its edges and service times of its nodes, i.e., $\sum_{e \in E(P_{uv})} l(e) + \sum_{v \in V(P_{uv})} g(v)$. Given a path $P = (s, v_2, v_3, \ldots, t)$ and a subset $S = \{v_{i_1}, v_{i_2} \ldots, v_{i_k}\}$ of $k \geq 1$ nodes in $V(P) \setminus \{s, t\}$ with $i_j < i_{j+1}$ for $j \in [k-1]$, we call $P[S]$ the *subpath of P induced by S* which is the path made of nodes $\{s, t\} \cup S$ and edges $\{\{s, v_{i_1}\}, \{v_{i_k}, t\}\} \cup \{\{v_{i_j}, v_{i_{j+1}}\} : j \in [k-1]\}$.

In the Capacitated Orienteering Problem (C-OP), we are given two distinguished nodes $s, t \in V$, a cost budget $B \in \mathbb{R}_{\geq 0}$ and a capacity bound $C \in \mathbb{R}_{\geq 0}$ on the sizes, and the goal is to find a path P_{st} from s to t in G that maximizes the prize $\pi(P_{st}) = \sum_{v \in V(P_{st})} \pi(v)$ and that satisfies both $l(P_{st}) + g(P_{st}) = \sum_{e \in E(P_{st})} l(e) + \sum_{v \in V(P_{st})} g(v) \leq B$ and $r(P_{st}) = \sum_{v \in V(P_{st})} r(v) \leq C$. W.l.o.g. we assume that $r(v) \leq C$, for any $v \in V$, and that $r(s) = r(t) = 0$. The Capacitated Team Orienteering Problem (C-TOP) is a generalization of C-OP in which we are asked to find $K \geq 1$ vertex-disjoint paths that maximize the total collected prize and each path respects both the capacity and the budget constraints. Formally, in C-TOP, the goal is to find K paths $P^1_{st}, \ldots, P^K_{st}$ from s to t that maximize $\sum_{k=1}^{K} \sum_{v \in P^k_{st}} \pi(v)$, and such that $l(P^k_{st}) + g(P^k_{st}) = \sum_{e \in E(P^k_{st})} l(e) + \sum_{v \in V(P^k_{st})} g(v) \leq B$ and $r(P^k_{st}) = \sum_{v \in V(P^k_{st})} r(v) \leq C$ for any $k \in [K]$.

In the remainder of the paper, we assume w.l.o.g. that the service times of s and t are equal to 0, that is $g(s) = g(t) = 0$. This implies that we can ignore the cost of service time by moving it to the edge length function. More formally, we redefine the length and service time functions as follows: the length is $l'(e) = l(e) + \frac{g(v)+g(u)}{2}$, for each edge $e = (u, v) \in E$, while the service time is $g'(v) = 0$ for each node $v \in V$. The cost of any path P_{st}, with the new functions, is therefore equal to $\sum_{e \in E(P_{st})} l'(e) = \sum_{e=(u,v) \in E(P_{st})} \left(l(e) + \frac{g(v)+g(u)}{2} \right) = \sum_{e \in E(P_{st})} l(e) + \sum_{v \in V(P_{st})} g(v)$. This implies that under this transformation: (1) the cost of any path is equal to the length of the path, and (2) clearly, the triangle inequality property is preserved. Thanks to this transformation, for the sake of simplicity and w.l.o.g., from now on we assume that any graph with node service times is converted to an equivalent graph with zero node service times. For the sake of readability, the obtained length l' will be denoted by l.

Given a subset of nodes $V' \subseteq V$, let $r(V') = \sum_{v \in V'} r(v)$ and $\pi(V') = \sum_{v \in V'} \pi(v)$. In all the benchmark instances that have been considered in the literature on C-OP, we observe that node size is positively correlated to node prize. In fact, in such real-world inspired instances, the prize of a node v is equal to $\pi(v) = (0.5 + h)r(v)$, where h is a random value in $[0, 1]$ (see [2, 25]). This implies that, for any two subsets of nodes $V_1, V_2 \subseteq V$ with $r(V_1) \geq r(V_2)$, we have $\pi(V_1) \geq \frac{1}{3}\pi(V_2)$, since $\pi(V_1) \geq \frac{1}{2}r(V_1)$ and $\pi(V_2) \leq \frac{3}{2}r(V_2) \leq \frac{3}{2}r(V_1)$. Indeed, in many practical applications we have that the prize of a subset of nodes increases as its size increases, that is for two subsets of nodes $V_1, V_2 \subseteq V$ with $r(V_1) \geq r(V_2)$, we have $\pi(V_1) \geq \lambda\pi(V_2)$, for some $\lambda \in (0, 1]$. Therefore, in this paper we consider C-OP and C-TOP under the following natural assumption.

Algorithm 1

```
Input: $I = \langle G = (V, E), s, t, \pi, l, B, r, C \rangle$.
Output: An $(s - t)$ path $P_{st}$ s.t. $l(P_{st}) \leq B$ and $r(P_{st}) \leq C$.
Let $I' = \langle G = (V, E), s, t, \pi, l, B \rangle$ be an instance of OP;
Apply an $A_{\text{OP}}$ to $I'$; let $P_\alpha$ be the returned solution;
if $r(P_\alpha) \leq C$ then $P_{st} \leftarrow P_\alpha$ ;
else                                                        // $r(P_\alpha) > C$
    Choose a subset of nodes $S \subseteq V(P_\alpha) \setminus \{s, t\}$ with $r(S) \geq C$ such that, for some $v \in S$,
     we have $r(S \setminus \{v\}) \leq C$;
    if $r(S) = C$ then $P_{st} \leftarrow P_\alpha[S]$;
    else                                                    // $r(S) > C$
        Let $v$ be a node in $S$ such that $r(S \setminus \{v\}) \leq C$;
        Partition $S$ into two subsets $S_1 = S \setminus \{v\}$ and $S_2 = \{v\}$;
        Let $S' = \arg\max_{A \in \{S_1, S_2\}} \pi(A)$;
        $P_{st} \leftarrow P_\alpha[S']$;
return $P_{st}$;
```

▶ **Assumption 2.1.** *Let $\lambda \in (0, 1]$ be a parameter to be fixed. For any two subsets of nodes $V_1, V_2 \subseteq V$ with $r(V_1) \geq r(V_2)$, we have $\pi(V_1) \geq \lambda\pi(V_2)$.*

Note that, Assumption 2.1 implies that selecting subsets of nodes with larger sizes results in collecting more prize, besides a multiplicative factor λ.

3 Approximation Algorithms with Theoretical Guarantees

In this section, we introduce some polynomial time algorithms for C-OP and C-TOP that guarantee bounded approximation ratios under Assumption 2.1. We first focus on C-OP under that assumption and introduce a polynomial time $\max\{\alpha, \frac{2}{\lambda}\}$-approximation algorithm where α is the approximation ratio guaranteed by an algorithm A_{OP} for OP that is used as a subroutine while $\lambda \in (0, 1]$ is the parameter of Assumption 2.1. Then, we show that this result implies, under some particular conditions, an improvement over the best known approximation ratio for C-OP. Finally, we show how to use this algorithm to approximate C-TOP.

Our main algorithm, whose pseudo-code is summarized in Algorithm 1, takes as input an instance $I = \langle G = (V, E), s, t, \pi, l, B, r, C \rangle$ of C-OP. Starting from I, Algorithm 1 defines an OP instance I' with $I' = \langle G = (V, E), s, t, \pi, l, B \rangle$ and executes algorithm A_{OP} onto it. Let P_α be the solution returned by A_{OP} when applied to I'. An optimal solution $OPT_{I'}$ to instance I' of OP has value at least $\pi(OPT_{I'}) \geq \pi(OPT_I)$, where OPT_I is an optimal solution to the instance I of C-OP. It follows that, if $r(P_\alpha) \leq C$, then P_α is an α-approximation also for I. Therefore, if $r(P_\alpha) \leq C$, Algorithm 1 returns P_α as a solution. If $r(P_\alpha) > C$, Algorithm 1 chooses a subset of nodes $S \subseteq V(P_\alpha) \setminus \{s, t\}$ such that $r(S) \geq C$ and, for some $v \in S$, we have $r(S \setminus \{v\}) \leq C$. Now if $r(S) = C$, then Algorithm 1 returns $P_\alpha[S]$, the subpath of P_α induced by S, as a solution. Otherwise, it partitions S into two subsets of nodes $S_1 = S \setminus \{v\}$ and $S_2 = \{v\}$, where v is a node in S such that $r(S_1) \leq C$. Note that $r(v) \leq C$ and hence both S_1 and S_2 are feasible solutions for I. Finally, Algorithm 1 selects the set with the maximum prize between S_1 and S_2, denoted by $S' = \arg\max_{A \in \{S_1, S_2\}} \pi(A)$, and returns $P_\alpha[S']$ as a solution. In the next theorem, we show that Algorithm 1 guarantees a $\max\{\alpha, \frac{2}{\lambda}\}$-approximation algorithm for C-OP under Assumption 2.1.

▶ **Theorem 1.** *Algorithm 1 is a polynomial time* $\max\{\alpha, \frac{2}{\lambda}\}$*-approximation algorithm for* C-OP *under Assumption 2.1, where* α *denotes the approximation ratio for* OP *and* $\lambda \in (0,1]$.

Proof. If $r(P_\alpha) \leq C$, then Algorithm 1 returns solution P_α. By the feasibility of P_α for instance I', we have that $l(P_\alpha) \leq B$ and hence P_α is feasible for instance I of C-OP. Moreover, $\pi(P_\alpha) \geq \frac{1}{\alpha}\pi(OPT_{I'}) \geq \frac{1}{\alpha}\pi(OPT_I)$, and hence P_α provides an α-approximation for I. If $r(P_\alpha) > C$, then Algorithm 1 selects a set $S \subseteq V(P_\alpha) \setminus \{s,t\}$ with $r(S) \geq C$ such that there exists a node $v \in S$ for which $r(S \setminus \{v\}) \leq C$. We distinguish between two cases.

1. $r(S) = C$. In this case, Algorithm 1 returns $P_\alpha[S]$ as a solution. Since $l(P_\alpha) \leq B$, then, by triangle inequality, we have $l(P_\alpha[S]) \leq B$. Moreover, $r(P_\alpha[S]) = r(S) = C$, as $r(s) = r(t) = 0$, and hence $P_\alpha[S]$ is feasible for I. By Assumption 2.1, it follows that $\pi(S) \geq \lambda\pi(OPT_I)$, since $r(S) = C$ and $r(OPT_I) \leq C$. Hence, $P_\alpha[S]$ is a $\frac{1}{\lambda}$-approximation for I.
2. $r(S) > C$. In this case, Algorithm 1 partitions S into two subsets $S_1 = S \setminus \{v\}$ and $S_2 = \{v\}$, with $r(S_1) \leq C$, selects the set with the maximum prize between S_1 and S_2, say S', and returns $P_\alpha[S']$ as solution. Since $l(P_\alpha) \leq B$, then, by triangle inequality, it follows that $l(P_\alpha[S']) \leq B$. Moreover, both $r(S_1)$ and $r(S_2)$ are upper bounded by C and hence $P_\alpha[S']$ is feasible for I. Regarding the approximation factor of $P_\alpha[S']$, we have $\pi(S') \geq \frac{1}{2}\pi(S) \geq \frac{\lambda}{2}\pi(OPT_I)$, where the first inequality holds as S' is the set with the maximum prize between two sets $S_1, S_2 \subseteq S$ with $S_1 \cup S_2 = S$ and $S_1 \cap S_2 = \emptyset$, and the second inequality follows by Assumption 2.1, as $r(S) \geq C$ and $r(OPT_I) \leq C$. Therefore, $P_\alpha[S']$ is a $\frac{2}{\lambda}$-approximation for I. ◀

Theorem 1, along with the $(2+\varepsilon)$-approximation algorithm for OP given by Chekuri et al. [9] implies the following result.

▶ **Corollary 2.** *For any fixed* $\varepsilon > 0$*, Algorithm 1 is a* $\max\{2+\varepsilon, \frac{2}{\lambda}\}$*-approximation algorithm for* C-OP*, under Assumption 2.1 where* $\lambda \in (0,1]$.

When $\lambda \geq \frac{2}{3}$ in Assumption 2.1, then $\frac{2}{\lambda} \leq 3$ and hence the above corollary implies that the approximation factor of Algorithm 1 is in the interval $[2+\varepsilon, 3]$, for any $\varepsilon \in (0,1]$. This is an improvement on the approximation of C-OP under Assumption 2.1 over the factor $3+\varepsilon$ by Bock and Sanità [8].

▶ **Corollary 3.** *For any fixed* $\varepsilon \in (0,1]$*, Algorithm 1 is a* β*-approximation algorithm with* $\beta \in [2+\varepsilon, 3]$ *for* C-OP*, under Assumption 2.1 where* $\lambda \in [\frac{2}{3}, 1]$.

Another interesting implication of Theorem 1 is that an α-approximation algorithm for OP results in an α-approximation for C-OP, under Assumption 2.1, when $\lambda \geq \frac{2}{\alpha}$. In particular, under this hypothesis, Algorithm 1 is a $(2+\varepsilon)$-approximation algorithm for C-OP, by using the result by Chekuri et al. [9].

▶ **Corollary 4.** *For any fixed* $\varepsilon > 0$*, Algorithm 1 is a* $(2+\varepsilon)$*-approximation algorithm for* C-OP*, under Assumption 1 where* $\lambda \geq \frac{2}{2+\varepsilon}$.

A β-approximation algorithm ALG for C-OP can be used as a black-box, to obtain a $(1 - e^{-\frac{1}{\beta}})^{-1}$-approximation algorithm for C-TOP [8], using the following greedy strategy (named GenStra):

1. For $i = 1$ to K do:
   a. Run the β-approximation algorithm for C-OP to obtain a path P_i on $G = (V,E)$.
   b. Remove all covered nodes $V(P_i)$ from G.
2. Return $P_1, \ldots, P_K$.

Using this result, we can generalize our result for C-OP to C-TOP under Assumption 2.1.

▶ **Theorem 5.** *Under Assumption 2.1, there exists a polynomial time* $(1-e^{-\frac{1}{\beta}})^{-1}$*-approximation algorithm for* C-TOP*, where* $\beta = \max\{2+\varepsilon, \frac{2}{\lambda}\}$ *for any fixed* $\varepsilon > 0$.

Proof. The theorem follows from Theorem 1 and the fact that any β-approximation algorithm for C-OP can be used as a subroutine in GenStra to achieve a $(1-e^{-\frac{1}{\beta}})^{-1}$-approximation factor for C-TOP [8]. ◀

Theorem 5 implies that when $\lambda \geq \frac{2}{3}$ in Assumption 2.1, one can achieve a ρ-approximation algorithm for C-TOP, where $\rho \in [(1-e^{-\frac{1}{2+\varepsilon}})^{-1}, (1-e^{-\frac{1}{3}})^{-1}]$, for any $\varepsilon \in (0,1)$, where $(1-e^{-\frac{1}{2}})^{-1} > 2.55$ and $(1-e^{-\frac{1}{3}})^{-1} < 3.53$. This is an improvement for C-TOP under Assumption 2.1 over the factor $(1-e^{-\frac{1}{3+\varepsilon}})^{-1}$, for $\varepsilon > 0$, by Bock and Sanità [8].

▶ **Corollary 6.** *For any fixed* $\varepsilon \in (0,1)$*, under Assumption 2.1 with* $\lambda > \frac{2}{3}$*,* C-TOP *admits a* ρ*-approximation algorithm, where* $\rho \in [2.55, 3.53)$.

4 Heuristic Algorithms

The running time of Algorithm 1 presented in Section 3 is dominated by the time required to run an approximation algorithm for OP at line 2. If we use the $(2+\varepsilon)$-approximation algorithm by Chekuri et al. [9] for this purpose, this step requires $O(n^{O(1/\varepsilon^2)})$ time, for any $\varepsilon > 0$. In this section, motivated by such high computational time, we design four efficient heuristic algorithms that have low computational time but do not guarantee any bound on the quality of the computed solution. In Section 5, we experimentally evaluate the proposed heuristics on relevant sets of instances of C-TOP, showing that they also produce high-quality solutions. In particular we show that our heuristics require small computational time and that the value of the computed solutions is comparable to that achieved by state-of-the-art methods. Both in this section and in Section 5, we assume that $s = t$ in C-OP and C-TOP, that is we need to find a tour instead of a path. We refer to node s as *depot*.

In what follows, we describe our heuristics for C-OP. Each algorithm ALG for C-OP can be generalized to be used for C-TOP by applying GenStra stated in Section 3, where we use ALG instead of a β-approximation algorithm for C-OP, and we set $s = t$.

Our heuristic algorithms for C-OP exploit a procedure, named DPROC, to produce solutions that respect the capacity constraints starting from a set of nodes $S \subseteq V$. Such procedure works as follows: first, it computes a subset of nodes $S' \subseteq S$ that maximizes the prize $\pi(S')$ and has size at most $r(S') \leq C$ by using the well-known dynamic programming for the Knapsack problem [27]. Then, it determines a tour T that includes the depot s and all nodes in S' using an approximation algorithm for the Traveling Salesman Problem (TSP). Specifically, for all algorithms we consider two versions of DPROC, which use either the 3/2- or 2-approximation for TSP [27], respectively, and, in Section 5, we will specify how the two versions are used in the experiments. Finally, DPROC returns T as output. We remark that the input graph is complete and metric. In the following, we denote the application of procedure DPROC with input S by DPROC(S). Now, for any two nodes u and v, let $w(u,v) = l(e) \cdot r(v)$, be the *weight* of edge $e = (u,v)$. Our heuristic algorithms for C-OP are as follows. In Section 5, we will extend each algorithm for C-OP to C-TOP by using procedure GenStra and, we call its extension with the same name for C-OP.

sqrB-ApxA (SBAA). This algorithm is inspired by the algorithms given by Kuo et al. [21] and by D'Angelo et al. [12] for the problem of finding a rooted out-tree in a directed graph that maximizes the sum of prizes associated to the nodes, subject to a budget constraint. Specifically, for each node $u \in V$, we compute a candidate set S_u and, at the end of the algorithm, we consider a set S_M that maximizes the prize, i.e. $S_M := \arg\max_{u\in V} \pi(S_u)$ and output $\textsc{dproc}(S_M)$. In details, the candidate set S_u of a node $u \in V$ is computed as follows. We first sort all nodes $v \in V$ in non-increasing order of $\pi(v)/w(u,v)$ or $\pi(v)$. In Section 5 we will describe how the two sorting strategies are used in the experiments. Then, we consider two integers x and y and, for each pair $(x,y) \in \{0,1,2\} \times [50]$, we compute: (i) the set S_y^x containing the first $yB^{1-x/2}$ nodes in the ordering that have a distance at most $B^{x/2}$ from u; (ii) a tour $T_y^x = \textsc{dproc}(S_y^x)$ and check if $l(T_y^x) \leq B$. Then, set S_u is selected as a set that produces a feasible tour in the previous step and maximizes the prize, i.e. $S_u := \arg\max\{\pi(S_y^x) : l(T_y^x) \leq B, (x,y) \in \{0,1,2\} \times [50]\}$. To improve the running time, we exploit the monotonicity of function π, iterate through the values of y from $y = 50$ to $y = 1$ and stop as soon as we find a feasible tour. The values for x and y have been chosen after a preliminary pilot experimental study on the algorithm's performance.

4-ApxA (4AA). This heuristic is based on the idea of Gupta et al. [18] who showed that, given an α-approximation algorithm for OP, it is possible to derive a 2α-approximation algorithm for C-OP. So, we use the best approximation algorithm for the unrooted version of OP in which there is no specified root node s that must be spanned, which is the 2-approximation algorithm proposed by Paul et al. [23]. In particular, given an instance $I = \langle G = (V,E), s, \pi, r, l, B, C\rangle$ of C-OP, we define an OP instance I' with $I' = \langle G = (V,E), s, \pi', l, B\rangle$ in which for any $v \in V$, $\pi'(v) = \pi(v) - \eta r(v)$, where $\eta \geq OPT/(2C)$ and OPT is an optimal solution to C-OP. As OPT is not known, we guess it through a binary search over the range $[\pi_{\min}, TP]$, where $\pi_{\min}$ be the minimum positive prize of a node and TP is the total prize of vertices. We know that $OPT \leq TP$. We estimate the value of OPT by guessing N possible values, where N is the smallest integer for which $\pi_{\min}2^{N-1} \geq TP$. For the instances considered in Section 5, we set η using this binary search. For each η, we compute the solution returned by the 2-approximation algorithm by Paul et al. [23] on the obtained instance I' and we let S_η be the nodes in this solution. By definition of OP, set S_η satisfies the budgeted constraint but it is not guaranteed to satisfy the capacity. Therefore, we compute $T_\eta = \textsc{dproc}(S_\eta)$ to obtain a tour that satisfies the capacity constraint. Finally we output the tour T_M that maximizes the prize, i.e. $T_M := \arg\max\{\pi(T_\eta)\}$, where η is set based on the binary search to find OPT. Note that for any v, in case $\pi'(v) = \pi(v) - \eta r(v) < 1$, we set $\pi'(v) = 1$.

GreedyRandom-ApxA (GRA). This is a modification of the randomized algorithm proposed by Arora and Scherer [4]. The following randomized algorithm is repeated multiple times and the solution with best prize is selected (in the experiments we repeat for 10 times). We keep a solution S, initially equal to the empty set. For $3|V|$ times we repeat the following loop. We sample a node v uniformly at random and we check if $v \in S$. If so, we remove it from S. Otherwise, we add it to S. Then, we compute $T = \textsc{dproc}(S)$ and check if $l(T) \leq B$ and $\pi(T) > \pi(S)$. In the affirmative case, we set $S := V(T)$ and repeat the loop.

Greedy-ApxA (GA). Like for SBAA, we compute a candidate set S_u, for each node $u \in V$, we select a set S_M that maximizes the prize, i.e. $S_M := \arg\max_{u\in V} \pi(S_u)$, and output $\textsc{dproc}(S_M)$. For each node $u \in V$, the candidate set S_u is computed as follows. We first sort all nodes $v \in V$ in non-increasing order of $\pi(v)/w(u,v)$ or $\pi(v)$. In Section 5 we will specify the used sorting strategy. We initialize S_u as $S_u := \{u\}$. Then, we iterate over the nodes $v \in V \setminus \{u\}$, according to the sorting. At each iteration we check whether adding to S_u the next node v in the sorting induces a feasible solution with better prize of the current solution. Specifically, we compute $T = \textsc{dproc}(S_u \cup \{v\})$ and check whether $l(T) \leq B$ and $\pi(T) > \pi(S_u)$. In the affirmative case, we set $S_u := V(T)$ and iterate to the next node in the ordering.

Note that SBAA, 4AA, GA and GreedyRandom-ApxA are pseudo-poly algorithms as we use the well-known dynamic programming for the Knapsack problem. However, one can use the well-known FPTAS for the knapsack problem [27].

5 Experiments

In this section, we present and analyze the results of an extensive experimental evaluation on the performance of the heuristics proposed in Section 4. We design two experiments, named respectively comparison and scalability, with the objective of answering to different experimental questions.

The aim of experiment comparison (see Section 5.1), is comparing the performance of the four proposed heuristic algorithms against methods of the literature that are considered the state-of-the-art for c-top. Among them, based on the most recent experimental results on the problem (see [19]), we identify the most effective/competitive w.r.t. solution quality and running time, that is algorithms: VNS, TSF, TSA [2]; BiFFf and BiFFs [25]; VSS-Tb and VSS-SA [6]; HALNS [19]. We do not consider, instead, algorithms ADEPT-RD [22], SA-ILS [17], and LNS/NLNS [20] since they have been tested only on a subset of the benchmark instances and, in terms of performance, they are dominated by or comparable to HALNS [19]; Furthermore, ILS [16] provided the average results on each set instead of giving their result on each instance.

The aim of experiment scalability (see Section 5.2), is assessing the scalability properties of our newly introduced heuristics, i.e. to study how the performance of our heuristics change with the input size, and specifically if our algorithms can process larger instances than those that have been considered in past literature on the problem. For all experiments, we use implementations of the four heuristics of Section 4 we developed for the purpose. All our code is written in C++ (available at https://shorturl.at/bMYNb) and compiled with GCC 9.4.0 with optimization level $O3$; all our tests have been executed on a workstation equipped with an Intel© Xeon© processor, clocked at 2.30GHz, running Ubuntu Linux.

5.1 comparison Experiment

In this experiment, we test implementations of SBAA, 4AA, GRA, and GA on two publicly available datasets of benchmark inputs for c-top, defined in [2] and [25], respectively, derived from instances of TSP and considered reference instances for assessing the performance of algorithms for c-top.

Input Data. The details of such datasets, which we call small-case and large-case inputs, respectively, are summarized in what follows:

small-case: this set contains 130 instances (divided into three subsets, named sc-1, sc-2 and sc-3 and having 10, 90 and 30 instances, resp.) defined in [2] by suitably manipulating the instances given in [11] for TSP. The number of nodes of graphs in this set is $n \in \{51, 76, 101, 121, 151, 200\}$; instances are generated by considering different combinations of fleet size $K \in \{2, 3, 4, 10, 15, 20\}$, budget $B \in \{50, 75, 100, 160, 200, 230, 720, 1040\}$ and capacity $C \in \{50, 75, 100, 140, 160, 200\}$.

large-case: this set contains 130 instances (divided in three subsets, named lc-1, lc-2, and lc-3 and having 10, 90, and 30 instances, resp.) developed in [25] by modifying the inputs to the Periodic Vehicle Routing Problem of [24]. The number of nodes in this set is $n \in \{337, 361, 385, 433, 481, 529, 505, 577\}$ while other parameters are $K \in \{6, 7, 8, 14, 15, 16, 18, 20, 21, 22, 24\}$, $B \in \{100, 200, 400, 660, 668, 675, 683, 705, 713, 720\}$ and $C \in \{75, 150, 200, 330, 335, 340, 345, 350, 360, 365, 375\}$.

Each of the above instances, in what follows, is identified by a unique string following the format *base*-n-K-C-B, where *base* is the name of the original TSP instance from either [11] or [24], while n is the number of nodes, K is the number of vehicles, C is the capacity and B is the budget. Note that, for both small-case and large-case instances, the prize of each node v having size $r(v)$ is assigned to be equal to $\pi(v) = (h + 0.5)r(v)$, where h is a random number uniformly generated within interval $[0, 1]$. This implies that for any instance having capacity C and number of vehicles K, the optimum for the instance is upper bounded by $(h + 0.5)KC \leq \frac{3KC}{2}$ in c-top.

Executed Tests. For all mentioned inputs, we run all four heuristics and measure both running time (column t, in seconds) and solution quality (i.e. achieved *prize*, column p). We then compare observed measures with the results obtained, on the same instances, by reference methods of the literature mentioned above, as summarized in Tables 1–4.

Table 1 Results of experiment comparison for small-case inputs, subset sc-1.

Instance	GRA			SBAA			GA			4AA			VNS [2]			TSF [2]			TSA [2]			BiFFf [25]			BiFFs [25]			VSS-Tb [6]			VSS-SA [6]			HALNS [19]		
	p	t	g	p	t	g	p	t	g	p	t	g	p	t	g	p	t	g	p	t	g	p	t	g	p	t	g	p	t	g	p	t	g	p	t	g
03-101-15-200-200	**1409**	< 1	0.00	**1409**	< 1	0.00	**1409**	< 1	0.00	904	362	35.84	**1409**	< 1	0.00	**1409**	< 1	0.00	**1409**	< 1	0.00	**1409**	< 1	0.00	**1409**	< 1	0.00	**1409**	< 1	0.00	**1409**	2	0.00	**1409**	< 1	0.00
06-51-10-160-200	**761**	< 1	0.00	**761**	< 1	0.00	**761**	< 1	0.00	191	73	74.90	**761**	< 1	0.00	**761**	< 1	0.00	**761**	< 1	0.00	**761**	< 1	0.00	**761**	< 1	0.00	**761**	< 1	0.00	**761**	< 1	0.00	**761**	< 1	0.00
07-76-20-140-160	**1327**	< 1	0.00	**1327**	< 1	0.00	**1327**	< 1	0.00	1238	146	6.70	**1327**	< 1	0.00	**1327**	< 1	0.00	**1327**	< 1	0.00	**1327**	< 1	0.00	**1327**	< 1	0.00	**1327**	< 1	0.00	**1327**	1	0.00	**1327**	< 1	0.00
08-101-15-200-230	**1409**	< 1	0.00	**1409**	< 1	0.00	**1409**	< 1	0.00	916	391	34.98	**1409**	< 1	0.00	**1409**	< 1	0.00	**1409**	< 1	0.00	**1409**	< 1	0.00	**1409**	< 1	0.00	**1409**	< 1	0.00	**1409**	1	0.00	**1409**	< 1	0.00
09-151-10-200-200	2063	< 1	0.09	2064	< 1	0.04	2057	< 1	0.38	1586	1255	0.00	2064	3600	0.00	2061	163	0.00	2062	127	0.00	**2065**	2	0.00	**2065**	2	0.00	**2065**	39	0.00	**2065**	120	0.00	**2065**	1	0.00
10-200-20-200-200	**3048**	< 1	0.00	**3048**	< 1	0.00	**3048**	< 1	0.00	2828	4218	7.21	**3048**	< 1	0.00	**3048**	< 1	0.00	**3048**	< 1	0.00	**3048**	< 1	0.00	**3048**	< 1	0.00	**3048**	< 1	0.00	**3048**	11	0.00	**3048**	< 1	0.00
13-121-15-200-720	**1287**	< 1	0.00	**1287**	< 1	0.00	**1287**	< 1	0.00	**1287**	417	0.00	**1287**	< 1	0.00	**1287**	< 1	0.00	**1287**	< 1	0.00	**1287**	< 1	0.00	**1287**	< 1	0.00	**1287**	< 1	0.00	**1287**	2	0.00	**1287**	< 1	0.00
14-101-10-200-1040	**1710**	< 1	0.00	**1710**	< 1	0.00	**1710**	< 1	0.00	**1710**	< 1	0.00	**1710**	< 1	0.00	**1710**	< 1	0.00	**1710**	< 1	0.00	**1710**	< 1	0.00	**1710**	< 1	0.00	**1710**	< 1	0.00	**1710**	3	0.00	**1710**	< 1	0.00
15-151-15-200-200	**2159**	< 1	0.00	**2159**	< 1	0.00	**2159**	< 1	0.00	**2159**	1450	0.00	**2159**	< 1	0.00	**2159**	< 1	0.00	**2159**	< 1	0.00	**2159**	< 1	0.00	**2159**	< 1	0.00	**2159**	< 1	0.00	**2159**	7	0.00	**2159**	< 1	0.00
16-200-15-200-200	2965	< 1	0.13	2965	< 1	0.13	2966	< 1	0.10	2941	3829	0.94	2968	3600	0.03	2965	270	0.13	2967	377	0.06	-	-	-	-	-	-	**2969**	61	0.00	**2969**	254	0.00	**2969**	76	0.00

Observe that, for subsets of inputs sc-2, sc-3, lc-2 and lc-3, which have a large number of instances, we report a meaningful selection of the results of our tests, while full data will appear in a longer version of the paper. Besides running time and prize, for each algorithm A and for each instance, we report the gap g_A between the solution Sol_A computed by A and the best known solution for the instance, obtained by any of the algorithm in the set X of considered algorithms, i.e. $g_A = \frac{BKS - Sol_A}{BKS} \cdot 100$, where $BKS = \max_{A' \in X} Sol_{A'}$. Algorithms achieving the maximum solution quality, for each instance, are highlighted in bold. For the sake of fairness, we remark that all the considered algorithms from the literature have a randomized nature and have been evaluated by following a measurement strategy commonly referred to as *Time-To-Best*, which consists of: (i) running a given algorithm 10 times; (ii) selecting the run that performs best in terms of solution quality (prize); (iii) reporting solution quality and running time only of such run of the algorithm (see [19] and references therein). While this measurement strategy is reasonable when one compares only randomized solutions, it appears to be not well suited to be applied in comparisons that include deterministic algorithms, such as ours GA, SBAA or 4AA, which output the same solution even if they are executed multiple times. Indeed, a more empirically appropriate assessment strategy would require to measure, for randomized approaches, the sum of the running times of the all executions, since that represents the actual time the algorithm have to run to obtain the best solution, and compare such time with that of deterministic algorithms. Therefore, running times reported for algorithms from the literature might likely be underestimations of the actual average running time.

Note that, procedure dproc, used by all our heuristics, considers different possibilities for computing a tour on a subset of the nodes, namely the 3/2- and 2-approximation algorithms for TSP [27]. Moreover, heuristics SBAA and GA use two different node sorting strategies, based on the prize or on the ratio between prize and weight. After a preliminary experimental

Table 2 Excerpt of the results of experiment COMPARISON for subsets SC-2 (top) and SC-3 (bottom).

Subset sc-2																																	
Instance	GRA			SBAA			GA			4AA			VNS [2]			TSF [2]			TSA [2]			BiFFf [25]			BiFFs [25]			VSS-Tb [6]			VSS-SA [6]			HALNS [19]		
	p	t	g	p	t	g	p	t	g	p	t	g	p	t	g	p	t	g	p	t	g	p	t	g	p	t	g	p	t	g	p	t	g	p	t	g
03-101-4-100-100	510	< 1	4.13	523	< 1	1.69	516	< 1	3.00	501	257	5.82	529	963	0.56	531	317	0.18	529	357	0.56	531	7	0.18	**532**	42	0.00	**532**	27	0.00	**532**	12	0.00	**532**	21	0.00
06-51-4-100-100	450	< 1	5.49	470	< 1	2.48	472	< 1	2.07	388	21	19.50	481	135	0.20	**482**	25	0.00	**482**	26	0.00	**482**	< 1	0.00	**482**	< 1	0.00	**482**	< 1	0.00	**482**	2	0.00	**482**	2	0.00
07-76-4-100-100	510	< 1	1.72	510	< 1	2.11	514	< 1	1.34	451	107	13.43	**521**	342	0.00	**521**	25	0.00	514	26	1.34	518	< 1	0.57	**521**	< 1	0.00	**521**	< 1	0.00	**521**	2	0.00	**521**	2	0.00
08-101-4-100-100	514	< 1	2.06	523	< 1	1.69	516	< 1	3.00	501	107	5.82	529	963	0.56	531	317	0.18	529	357	0.56	531	7	0.18	**532**	41	0.00	**532**	29	0.00	**532**	20	0.00	**532**	31	0.00
09-151-4-100-100	532	< 1	1.64	542	< 1	0.73	539	< 1	1.26	506	1074	7.32	545	2934	0.18	539	924	1.28	536	959	1.83	545	51	0.18	**546**	38	0.00	**546**	53	0.00	**546**	31	0.00	**546**	31	0.00
10-200-4-100-100	544	< 1	1.80	548	< 1	0.90	550	< 1	0.54	522	2969	5.60	548	3600	0.90	549	2077	0.72	550	3232	0.54	**553**	183	0.00	**553**	243	0.00	**553**	11	0.00	**553**	40	0.00	**553**	43	0.00
13-121-4-100-100	415	< 1	1.19	415	< 1	0.95	417	< 1	0.47	383	23	8.59	**419**	179	0.00	**419**	24	0.00	**419**	48	0.00	**419**	< 1	0.00	**419**	< 1	0.00	**419**	< 1	0.00	**419**	1	0.00	**419**	1	0.00
14-101-4-100-100	511	< 1	3.04	522	< 1	0.57	521	< 1	0.76	488	212	7.04	**525**	670	0.00	523	210	0.38	**525**	292	0.00	**525**	< 1	0.00	**525**	< 1	0.00	**525**	5	0.00	**525**	1	0.00	**525**	5	0.00
15-151-4-100-100	542	< 1	1.27	545	< 1	0.72	544	< 1	0.91	518	1077	5.64	548	2828	0.18	**549**	1252	0.00	545	1015	0.72	548	10	0.18	**549**	206	0.00	**549**	66	0.00	**549**	49	0.00	**549**	87	0.00
16-200-4-100-100	553	< 1	0.53	555	< 1	0.53	554	< 1	0.71	538	3009	3.58	554	3600	0.71	554	2124	0.71	553	3559	0.89	556	3	0.35	**558**	67	0.00	**558**	5	0.00	**558**	79	0.00	**558**	36	0.00

Subset sc-3																																	
Instance	GRA			SBAA			GA			4AA			VNS [2]			TSF [2]			TSA [2]			BiFFf [25]			BiFFs [25]			VSS-Tb [6]			VSS-SA [6]			HALNS [19]		
	p	t	g	p	t	g	p	t	g	p	t	g	p	t	g	p	t	g	p	t	g	p	t	g	p	t	g	p	t	g	p	t	g	p	t	g
03-101-4-200-200	936	< 1	1.47	938	< 1	1.26	939	< 1	1.15	914	927	3.78	**950**	961	0.00	946	110	0.42	947	42	0.31	**950**	< 1	0.00	**950**	< 1	0.00	**950**	16	0.00	**950**	11	0.00	**950**	10	0.00
06-51-4-160-200	681	< 1	0.29	682	< 1	0.14	680	< 1	0.43	648	82	5.12	**683**	53	0.00	**683**	5	0.00	682	4	0.14	**683**	< 1	0.00	**683**	< 1	0.00	**683**	< 1	0.00	**683**	1	0.00	**683**	1	0.00
07-76-4-140-160	705	< 1	0.28	705	< 1	0.28	704	< 1	0.42	686	376	2.97	**707**	296	0.00	**707**	44	0.00	702	39	0.70	**707**	2	0.00	**707**	5	0.00	**707**	2	0.00	**707**	1	0.00	**707**	< 1	0.00
08-101-4-200-230	947	< 1	0.31	949	< 1	0.10	946	< 1	0.42	924	916	2.73	**950**	726	0.00	949	89	0.10	949	38	0.10	**950**	1	0.00	**950**	10	0.00	**950**	8	0.00	**950**	8	0.00	**950**	11	0.00
09-151-4-200-200	1029	< 1	0.38	1031	< 1	0.19	1024	4	0.87	1008	3552	2.42	**1033**	2903	0.00	**1033**	480	0.00	1029	254	0.38	**1033**	2	0.00	**1033**	2	0.00	**1033**	44	0.00	**1033**	11	0.00	**1033**	16	0.00
10-200-4-200-200	**1064**	< 1	0.00	**1064**	< 1	0.00	1062	10	0.18	1060	8854	0.37	**1064**	3600	0.00	**1064**	1260	0.00	1063	789	0.09	**1064**	1	0.00	**1064**	< 1	0.00	**1064**	15	0.00	**1064**	9	0.00	**950**	11	0.00
13-121-4-200-720	**908**	< 1	0.00	**908**	< 1	0.00	**908**	< 1	0.00	**908**	1474	0.00	**908**	954	0.00	907	76	0.11	906	40	0.22	**908**	< 1	0.00	**908**	< 1	0.00	**908**	217	0.00	**908**	27	0.00	**908**	6	0.00
14-101-4-200-1040	**975**	< 1	0.00	**975**	< 1	0.00	**975**	< 1	0.00	**978**	< 1	0.00	**975**	483	0.00	**975**	56	0.00	975	38	0.00	**975**	1	0.00	**975**	1	0.00	**975**	< 1	0.00	**975**	1	0.00	**975**	< 1	0.00
15-151-4-200-200	1024	< 1	0.67	1027	< 1	0.38	1016	5	1.45	1010	3645	2.03	**1031**	2832	0.00	1019	618	1.16	1030	276	0.09	**1031**	3	0.00	**1031**	3	0.00	**1031**	262	0.00	**1031**	58	0.00	**1031**	7	0.00
16-200-4-200-200	1071	< 1	0.18	1071	< 1	0.18	1068	11	0.46	1062	9171	1.02	**1073**	3600	0.00	1072	1263	0.09	1071	897	0.18	**1073**	1	0.00	**1073**	1	0.00	**1073**	40	0.00	**1073**	15	0.00	**1073**	1	0.00

study, we found out that on instances SC-1, SC-2, and SC-3, on average the algorithms based on prize ordering performs worse in terms of collected prize than those based on prize-over-weight ordering. Therefore, for these instances we use the prize-over-weight ordering and both the 3/2- and 2-approximation algorithms for TSP. We then select the solution with the highest prize between these two and report the sum of the running times of both approaches. Similarly, for instances LC-1, LC-2 and LC-3, our preliminary experiments show that algorithms based on the 2-approximation algorithm for TSP perform worse than those based on the 3/2-approximation and hence, in these instances, we use only this latter and both prize and prize-over-weight orders. We then select the solution with the highest prize and report the sum of the running times of both approaches. Finally, for SBAA we fix parameter y, determining an upper bound on how many nodes can be assigned to each vehicle, to 20, whenever the number of vehicles is large enough so that the nodes of graphs can be divided among vehicles.

Analysis. Our data lead to two main general conclusions: first, the newly introduced algorithms are competitive with existing ones in terms of solution quality. In fact, they achieve, in many cases, best known solutions (i.e. have zero gap), and solutions with good quality, with gaps that are in the order of few percentage points, one or two tens in the worst cases, in the remaining cases. Second, SBAA, GA and GRA are significantly faster than methods known in the literature, requiring running times that are up to orders of magnitude smaller to achieve solutions that have comparable quality (with prizes equal or very close to the best ones and corresponding small gaps). The only exception to this behavior is algorithm 4AA, whose running time does not scale well with the graph size, due to using the 2-approximation algorithm of [23]. For this reason, we omit from the comparison the results of algorithm 4AA for larger instances, i.e. LARGE-CASE. In more details, we observe that:

Table 3 Results of experiment COMPARISON for LARGE-CASE inputs, subset LC-1.

Instance	GRA			SBAA			GA			BiFFf [25]			BiFFs [25]			VSS-Tb [6]			VSS-SA [6]			HALNS [19]		
	p	t	g	p	t	g	p	t	g	p	t	g	p	t	g	p	t	g	p	t	g	p	t	g
01-337-14-345-720	3836	30	8.05	4139	1	0.79	4014	51	3.78	**4172**	17	0.00	**4172**	17	0.00	**4172**	1	0.37	**4172**	2	0.00	**4172**	1	0.00
02-385-16-350-713	4434	43	7.31	4753	< 1	0.64	4605	77	3.74	**4784**	25	0.00	**4784**	25	0.00	**4784**	1	0.37	**4784**	3	0.00	**4784**	2	0.00
03-433-18-330-675	4911	59	5.57	5187	< 1	0.26	4949	103	4.84	**5201**	33	0.00	**5201**	32	0.00	**5201**	2	0.37	**5201**	3	0.00	**5201**	4	0.00
04-481-20-335-713	5463	85	6.35	**5834**	2	0.00	5697	155	2.24	5828	48	0.10	5828	47	0.10	5828	1	0.10	5828	3	0.10	5828	3	0.10
05-529-22-340-705	5892	127	8.59	**6446**	5	1.95	6211	215	3.63	6445	101	0.01	6445	98	0.01	6445	3	0.01	6445	5	0.01	6445	3	0.01
06-577-24-365-683	6709	164	5.26	**7082**	5	0.00	6937	266	1.89	7071	94	0.15	7071	93	0.15	7071	1	0.15	7071	2	0.15	7071	6	0.15
07-361-15-335-668	3877	41	10.97	4134	< 1	5.07	4122	56	5.35	**4355**	24	0.00	**4355**	23	0.00	**4355**	1	0.37	**4355**	3	0.00	**4355**	1	0.00
08-433-18-350-675	4706	74	9.39	5133	2	1.17	4965	122	4.40	**5194**	51	0.00	**5194**	49	0.00	**5194**	2	0.37	**5194**	4	0.00	**5194**	1	0.00
09-505-21-360-660	5267	118	14.81	5841	5	5.53	5953	176	3.71	**6183**	103	0.00	**6183**	104	0.00	**6183**	3	0.37	**6183**	22	0.00	**6183**	5	0.00
10-577-24-375-675	6601	166	8.88	**7245**	8	0.00	7132	289	1.47	7239	144	0.08	7239	144	0.08	7239	4	0.08	7239	7	0.08	7239	6	0.08

- for SMALL-CASE instances, algorithms GRA, SBAA and GA outperform all other approaches in terms of running time by completing their execution always in less than 1 second; at the same time they compute best solutions in all cases with few exceptions where the gap is below 5%; in more details, for subset SC-1, algorithms VSS-Tb, VSS-SA and HALNS have running times up to 254 seconds (with zero gap) while GRA and SBAA, GA take less than 1 second (with gaps below 0.39%); for subset SC-2, similarly, VSS-Tb, VSS-SA and HALNS have running times up to 100 seconds (with zero gap) while our simple algorithms SBAA and GRA take always less than 1 second (with gaps below 5% and 7%, resp.); instead, GA has running time below 4 seconds (while exhibiting gaps below 5%); finally, for subset SC-3, VSS-Tb, VSS-SA and HALNS have running times up to 300 seconds, while SBAA and GRA run always for less than 1 second and their gaps are below 1.78% and 3.42%, resp.; GA has running time up to 11 seconds with gap below 2.00%; Note that for subsets SC-1, SC-2 and SC-3, 4AA has gap mostly below 15.00% with high running time.

- for LARGE-CASE instances, algorithm SBAA outperforms the literature in 4 out of 10 instances of subset LC-1, in terms of quality of solution, while having running time at most 8 seconds; in the remaining 6 instances of subset LC-1, method SBAA is competitive w.r.t. the state-of-the-art, in terms of quality of solution, while achieving a gap that is always below 5.40%; for subset LC-2, algorithms VSS-Tb, VSS-SA and HALNS have large running times (up to 3900 seconds) while SBAA and GRA are the best performing in terms of time, with executions lasting at most 32 seconds (which is at least two orders of magnitude faster than VSS-Tb, VSS-SA and HALNS); on top of that, the gap obtained by SBAA and GRA remain below 15% and 20% respectively in most cases, and the gap of GA is mostly below 15% (with running time up to 132 seconds); finally, for subset LC-3, algorithms VSS-Tb, VSS-SA and HALNS have huge running times (up to 16000 and 1700 seconds, resp., for VSS-Tb and VSS-SA, while HALNS runs for up to 7000 seconds) while SBAA and GRA run for at most 56 seconds (meaning that SBAA is at least two orders of magnitude faster than VSS-Tb, VSS-SA and HALNS) with gap mostly below 8%; similarly, GRA has running time up to 56 seconds with the gap mostly below 15%, and GA takes up to 437 seconds to yield gaps that are mostly below 5%; to summarize, the results for LARGE-CASE inputs suggest that our very simple algorithms outperform the literature by far in terms of time (at least an order of magnitude) while having a good gap, and hence they can be considered more practical.

Table 4 Excerpt of the results of experiment COMPARISON for subsets LC-2 (top) and LC-3 (bottom).

Subset lc-2																								
Instance	GRA			SBAA			GA			BiFFf [25]			BiFFs [25]			VSS-Tb [6]			VSS-SA [6]			HALNS [19]		
	p	t	g	p	t	g	p	t	g	p	t	g	p	t	g	p	t	g	p	t	g	p	t	g
81-337-8-200-400	1718	16	15.82	1911	16	6.36	1976	50	3.57	2032	155	0.44	2032	1236	0.44	2039	1286	0.10	**2041**	763	0.00	2040	1167	0.05
81-337-8-200-400	1757	8	13.91	1911	16	6.36	1976	50	3.57	2032	155	0.44	2032	1236	0.44	2039	1286	0.10	**2041**	763	0.00	2040	1167	0.05
82-385-8-200-400	1830	8	11.50	1850	15	10.54	2022	55	2.22	2064	1216	0.19	2065	5641	0.15	2066	2121	0.10	**2068**	1078	0.00	2066	1983	0.10
83-433-8-200-400	1939	16	7.57	2024	10	3.52	2058	77	1.90	2096	574	0.10	2097	3173	0.05	2097	2724	0.05	**2098**	1649	0.00	2098	1562	0.00
84-481-8-200-400	1876	16	11.96	2014	21	5.49	2075	106	2.62	2127	112	0.19	2127	103	0.19	2130	3317	0.05	**2131**	1627	0.00	2130	1116	0.05
85-529-8-200-400	1708	20	21.03	2023	24	6.47	2109	111	2.77	2154	1038	0.32	2155	7914	0.37	2162	3764	0.05	**2163**	2252	0.00	2161	1339	0.09
86-577-8-200-400	1991	24	9.66	2116	19	3.99	2171	132	1.54	2204	7385	0.05	2204	1658	0.05	**2205**	6426	0.00	**2204**	2289	0.05	**2205**	2078	0.00
87-361-8-200-400	1804	8	12.59	1939	16	6.05	1978	55	4.16	2063	1964	0.05	2063	3762	0.05	2063	889	0.05	**2064**	886	0.00	**2064**	1329	0.00
88-433-8-200-400	1800	16	13.21	1814	22	12.53	2011	90	3.03	2068	683	0.29	2070	3589	0.19	2072	1974	0.10	**2074**	2000	0.00	2072	784	0.10
89-505-8-200-400	1718	16	18.88	1942	32	8.30	2051	126	3.03	2115	11022	0.14	2115	17380	0.14	2116	2693	0.09	**2118**	2515	0.00	2115	2849	0.14
90-577-8-200-400	1961	24	9.75	2077	24	4.41	2139	129	1.56	2168	2168	0.23	2172	15678	0.05	2171	3326	0.09	**2173**	3197	0.00	**2173**	2901	0.00

Subset lc-3																								
Instance	GRA			SBAA			GA			BiFFf [25]			BiFFs [25]			VSS-Tb [6]			VSS-SA [6]			HALNS [19]		
	p	t	g	p	t	g	p	t	g	p	t	g	p	t	g	p	t	g	p	t	g	p	t	g
21-337-8-345-720	2858	19	9.52	2933	26	7.15	2975	135	5.82	**3159**	927	0.00	**3159**	1745	0.00	3158	1462	0.03	**3159**	1576	0.00	**3159**	824	0.00
22-385-8-350-713	2911	24	11.54	3033	26	7.83	3183	161	3.28	3290	144	0.03	**3291**	583	0.00	**3291**	2850	0.03	**3291**	3018	0.00	**3291**	1401	0.00
23-433-8-330-675	2970	26	6.89	3058	17	4.13	3143	188	1.47	**3190**	455	0.00	**3190**	461	0.00	**3190**	4288	0.00	**3190**	3143	0.00	**3190**	616	0.00
24-481-8-335-713	3038	34	7.63	3144	29	4.40	3218	273	2.15	**3289**	451	0.00	**3289**	426	0.00	**3289**	283	0.00	**3289**	314	0.00	**3289**	297	0.00
25-529-8-340-705	3078	41	10.36	3258	31	5.12	3368	295	1.92	3432	1953	0.06	**3434**	7418	0.00	**3434**	4145	0.00	**3434**	4396	0.00	**3434**	5964	0.00
26-577-8-365-683	3385	51	9.70	3538	33	3.70	3674	406	2.00	**3749**	997	0.00	**3749**	1007	0.00	3748	14090	0.03	3748	9342	0.03	**3749**	1430	0.00
27-361-8-335-668	2738	24	12.15	3916	22	6.44	2947	131	5.45	3116	187	0.03	3116	189	0.03	**3117**	3090	0.00	**3117**	2248	0.00	**3117**	2046	0.00
28-433-8-350-675	2828	33	14.38	3106	40	5.96	3166	235	4.14	3301	1834	0.06	3302	3455	0.03	**3303**	5691	0.00	**3303**	5120	0.00	**3303**	3583	0.00
29-505-8-360-660	2794	45	20.82	3083	55	12.63	3303	373	6.40	3510	2282	0.54	3525	14499	0.11	3528	7370	0.03	**3529**	11119	0.00	3526	7134	0.09
30-577-8-375-675	3233	56	14.53	3518	48	7.00	3698	437	2.24	3781	2784	0.05	**3783**	11663	0.00	3781	10154	0.05	**3783**	9865	0.00	**3783**	7147	0.00

5.2 scalability Experiment

Here we evaluate how the running times of SBAA, GRA and GA change as the input size increases.

Input Data. We generate input instances whose size is far larger than that of any of the available benchmark inputs, with up to 15 500 nodes, by manipulating instance BRUSSELS2, used by Arnold et al. [3], for a version of the capacitated vehicle routing problem where, given a graph with edge lengths and a set of vehicles with limited capacity, the goal is to cover all the nodes with minimum total length and in such a way that each vehicle respects the capacity constraint. We sample uniformly at random 10 subgraphs from BRUSSELS2, each having from 500 nodes to 15 500 nodes with steps of 1 000 nodes. For each subgraph, we consider $K \in \{2, 4, \dots, 10\}$ and $B \in \{200, 400, 600\}$ while the capacity is fixed to $C = 200$. Note that in the original instance, BRUSSELS2, the capacity of each vehicle is set to 150. Similarly to the other benchmark instances, the prize is set to $\pi(v) = (h + 0.5)r(v)$, for each node v with size $r(v)$, with h randomly uniformly chosen within $[0, 1]$.

Executed Tests. For each subgraph and combination of K and B, we ran heuristics `SBAA`, `GRA`, and `GA`, and measure achieved prize and running time. We omit heuristic `4AA` from this part of the study since its running time is observed to be high even for not so large input combinations (see Section 5.1). The results of this experiment are summarized in Figures 1– 3: for each heuristic and for each considered value of B, we report measured solution quality and running time, averaged over all K.

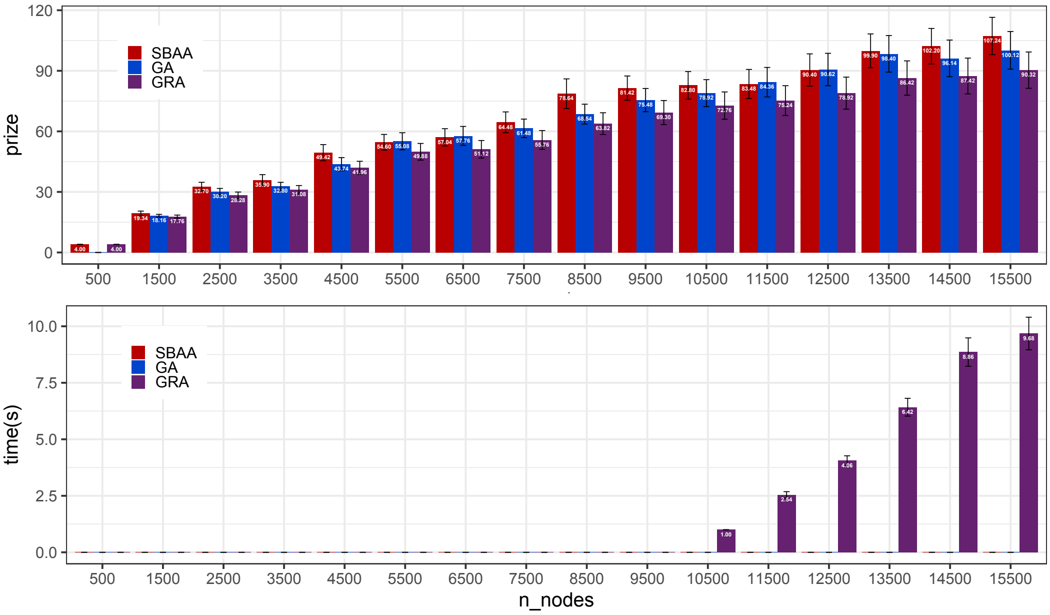


Figure 1 Results of the SCALABILITY experiment for $B = 200$.

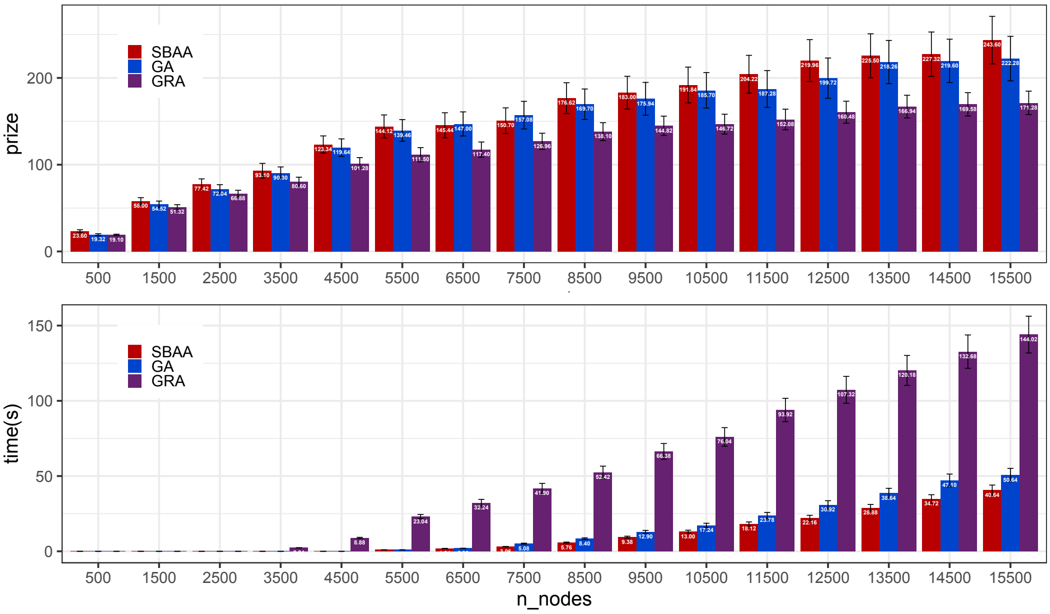


Figure 2 Results of the SCALABILITY experiment for $B = 400$.

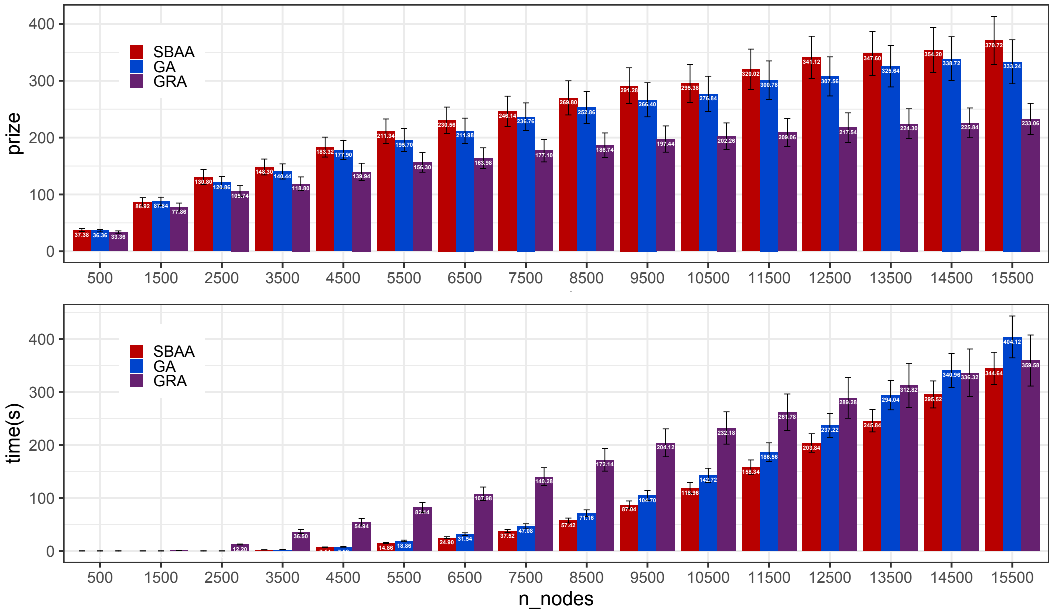


Figure 3 Results of the SCALABILITY experiment for $B = 600$.

Analysis. Our experimental data highlight the following general behavior. When $B = 200$ (see Figure 1), GA and SBAA are extremely fast, with running times smaller than 1 second even when the number of nodes n approaches 15 500; the running time of GRA is higher than the first two heuristics and grows faster as n increases, but remains below 10 seconds even for the largest case of $n = 15\,500$. The latter value is far below the average running times of algorithms tested in Section 5.1 for smaller graphs. Moreover, despite the low running time, GA and SBAA on average outperform GRA also w.r.t. achieved prize. When the budget is increased to 400 (see Figure 2) the observed trends are similar, with the average running time of GA and SBAA being below 1 second until n is less than 6 500 and remaining below 50 seconds. GA and SBAA outperform GRA w.r.t. both execution time and prize while SBAA outperforms GA, by small factors, w.r.t. both execution time and prize. Finally, when B is further increased to 600 (Figure 3), the trend in terms of solution quality appears not to be affected while the running time of all considered heuristics significantly increases, with the difference between SBAA, GA and GRA that seems to decrease as n approaches the largest value of 15 000. In general, our experiments suggest that the running time of SBAA and GA tends to grow approximately linearly with the input size and highlight that, on the the largest instance, SBAA and GA take below one minute on average, whereas previous algorithms are not able to handle such large input graphs. Note that however, we use the dynamic programming for the knapsack problem in both SBAA and GA, the capacity C in our instances is less than the number of nodes.

References

1 Claudia Archetti, Nicola Bianchessi, and Maria Grazia Speranza. Optimal solutions for routing problems with profits. *Discret. Appl. Math.*, 161(4-5):547–557, 2013.

2 Claudia Archetti, Dominique Feillet, Alain Hertz, and Maria Grazia Speranza. The capacitated team orienteering and profitable tour problems. *J. Oper. Res. Soc.*, 60(6):831–842, 2009.

3 Florian Arnold, Michel Gendreau, and Kenneth Sörensen. Efficiently solving very large-scale routing problems. *Comput. Oper. Res.*, 107:32–42, 2019.

4 Sankalp Arora and Sebastian A. Scherer. Randomized algorithm for informative path planning with budget constraints. In *2017 IEEE International Conference on Robotics and Automation, ICRA 2017, Singapore, Singapore, May 29 - June 3, 2017*, pages 4997–5004. IEEE, 2017.

5 Nikhil Bansal, Avrim Blum, Shuchi Chawla, and Adam Meyerson. Approximation algorithms for deadline-tsp and vehicle routing with time-windows. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 166–174. ACM, 2004.

6 Asma Ben-Said, Racha El-Hajj, and Aziz Moukrim. A variable space search heuristic for the capacitated team orienteering problem. *J. Heuristics*, 25(2):273–303, 2019.

7 Avrim Blum, Shuchi Chawla, David R. Karger, Terran Lane, Adam Meyerson, and Maria Minkoff. Approximation algorithms for orienteering and discounted-reward TSP. *SIAM J. Comput.*, 37(2):653–670, 2007.

8 Adrian Bock and Laura Sanità. The capacitated orienteering problem. *Discret. Appl. Math.*, 195:31–42, 2015.

9 Chandra Chekuri, Nitish Korula, and Martin Pál. Improved algorithms for orienteering and related problems. *ACM Trans. Algorithms*, 8(3):23:1–23:27, 2012.

10 Ke Chen and Sariel Har-Peled. The euclidean orienteering problem revisited. *SIAM J. Comput.*, 38(1):385–397, 2008.

11 Nicos Christofides. The vehicle routing problem. *Revue française d'automatique, informatique, recherche opérationnelle. Recherche opérationnelle*, 10(V1):55–70, 1976.

12 Gianlorenzo D'Angelo, Esmaeil Delfaraz, and Hugo Gilbert. Budgeted out-tree maximization with submodular prizes. In Sang Won Bae and Heejin Park, editors, *33rd International Symposium on Algorithms and Computation, ISAAC 2022, December 19-21, 2022, Seoul, Korea*, volume 248 of *LIPIcs*, pages 9:1–9:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

13 Mattia D'Emidio, Esmaeil Delfaraz, Gabriele Di Stefano, Giannantonio Frittella, and Edgardo Vittoria. Route planning algorithms for fleets of connected vehicles: State of the art, implementation, and deployment. *Applied Sciences*, 14(7), 2024. doi:10.3390/app14072884.

14 Zachary Friggstad, Sreenivas Gollapudi, Kostas Kollias, Tamás Sarlós, Chaitanya Swamy, and Andrew Tomkins. Orienteering algorithms for generating travel itineraries. In Yi Chang, Chengxiang Zhai, Yan Liu, and Yoelle Maarek, editors, *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, WSDM 2018, Marina Del Rey, CA, USA, February 5-9, 2018*, pages 180–188. ACM, 2018.

15 Zachary Friggstad and Chaitanya Swamy. Compact, provably-good lps for orienteering and regret-bounded vehicle routing. In Friedrich Eisenbrand and Jochen Könemann, editors, *Integer Programming and Combinatorial Optimization - 19th International Conference, IPCO 2017, Waterloo, ON, Canada, June 26-28, 2017, Proceedings*, volume 10328 of *Lecture Notes in Computer Science*, pages 199–211. Springer, 2017.

16 Aldy Gunawan, Kien Ming Ng, Vincent F Yu, Gordy Adiprasetyo, and Hoong Chuin Lau. The capacitated team orienteering problem. *Proceedings of the 9th International Conference on Industrial Engineering and Operations ManagementBangkok, Thailand, March 5-7, 2019*, pages 1630–1638, 2019.

17 Aldy Gunawan, Jiahui Zhu, and Kien Ming NG. The capacitated team orienteering problem: a hybrid simulated annealing and iterated local search approach. In *Proceedings of the 13th International Conference on the Practice and Theory of Automated Timetabling-PATAT*, volume 2, 2021.

18 Anupam Gupta, Ravishankar Krishnaswamy, Viswanath Nagarajan, and R. Ravi. Running errands in time: Approximation algorithms for stochastic orienteering. *Math. Oper. Res.*, 40(1):56–79, 2015.

19 Farouk Hammami. An efficient hybrid adaptive large neighborhood search method for the capacitated team orienteering problem. *Expert Systems with Applications*, 249:123561, 2024.

20 André Hottung and Kevin Tierney. Neural large neighborhood search for routing problems. *Artif. Intell.*, 313:103786, 2022.

21 Tung-Wei Kuo, Kate Ching-Ju Lin, and Ming-Jer Tsai. Maximizing submodular set function with connectivity constraint: Theory and application to networks. *IEEE/ACM Trans. Netw.*, 23(2):533–546, 2015.

22 Zhixing Luo, Brenda Cheang, Andrew Lim, and Wenbin Zhu. An adaptive ejection pool with toggle-rule diversification approach for the capacitated team orienteering problem. *Eur. J. Oper. Res.*, 229(3):673–682, 2013.

23 Alice Paul, Daniel Freund, Aaron M. Ferber, David B. Shmoys, and David P. Williamson. Budgeted prize-collecting traveling salesman and minimum spanning tree problems. *Math. Oper. Res.*, 45(2):576–590, 2020.

24 Sandro Pirkwieser and Günther R. Raidl. Multilevel variable neighborhood search for periodic routing problems. In Peter I. Cowling and Peter Merz, editors, *Evolutionary Computation in Combinatorial Optimization, 10th European Conference, EvoCOP 2010, Istanbul, Turkey, April 7-9, 2010. Proceedings*, volume 6022 of *Lecture Notes in Computer Science*, pages 226–238. Springer, 2010.

25 Christos D. Tarantilis, Foteini Stavropoulou, and Panagiotis P. Repoussis. The capacitated team orienteering problem: A bi-level filter-and-fan method. *Eur. J. Oper. Res.*, 224(1):65–78, 2013.

26 Dimitra Trachanatzi, Manousos Rigakis, Andromachi Taxidou, Magdalene Marinaki, Yannis Marinakis, and Nikolaos F. Matsatsinis. A novel solution encoding in the differential evolution algorithm for optimizing tourist trip design problems. In Nikolaos F. Matsatsinis, Yannis Marinakis, and Panos M. Pardalos, editors, *Learning and Intelligent Optimization - 13th International Conference, LION 13, Chania, Crete, Greece, May 27-31, 2019, Revised Selected Papers*, volume 11968 of *Lecture Notes in Computer Science*, pages 253–267. Springer, 2019.

27 Vijay V Vazirani. *Approximation algorithms*, volume 1. Springer, 2001.

28 Wenzheng Xu, Weifa Liang, Zichuan Xu, Jian Peng, Dezhong Peng, Tang Liu, Xiaohua Jia, and Sajal K. Das. Approximation algorithms for the generalized team orienteering problem and its applications. *IEEE/ACM Trans. Netw.*, 29(1):176–189, 2021.

29 Wenzheng Xu, Chengxi Wang, Hongbin Xie, Weifa Liang, Haipeng Dai, Zichuan Xu, Ziming Wang, Bing Guo, and Sajal K Das. Reward maximization for disaster zone monitoring with heterogeneous uavs. *IEEE/ACM Transactions on Networking*, 2023.

New Bounds on the Performance of SBP for the Dial-a-Ride Problem with Revenues

Barbara M. Anthony
Southwestern University, Georgetown, TX, USA

Christine Chung
Connecticut College, New London, CT, USA

Ananya Das
Middlebury College, VT, USA

David Yuen
Independent Researcher, Kapolei, HI, USA

Abstract

We revisit the Segmented Best Path (SBP) algorithm for online DARP in an offline setting with revenues and a time limit. The goal is to find a subset of the inputted ride requests that can be served within the time limit while maximizing the total revenue earned. SBP divides the time into segments and greedily chooses the highest-revenue path of requests to serve within each time segment. We show that SBP's performance has an upper bound of 5. Further, while SBP is a tight 4-approximation in the uniform-revenue case, we find that with non-uniform revenues, the approximation ratio of SBP has a lower bound strictly greater than 4; in particular, we provide a lower bound of $(\sqrt{e}+1)/(\sqrt{e}-1) \approx 4.08299$, which we show can be generalized to instances with ratio greater than 4.278.

2012 ACM Subject Classification Theory of computation → Routing and network design problems; Applied computing → Transportation

Keywords and phrases Dial-a-Ride problems, Lower bounds, Vehicle routing

Digital Object Identifier 10.4230/OASIcs.ATMOS.2024.8

Category Short Paper

1 Introduction

We study the Dial-a-Ride Problem in an offline setting with revenues and a time limit T. The goal is to find a subset of the inputted ride requests that can be served within the time limit while maximizing the total revenue earned. We consider the Segmented Best Path (SBP) algorithm, originally proposed in [6] for an online variant of DARP. It was later adapted by [1] for the offline setting where revenues are uniform and the goal is to maximize the number of requests served. We present SBP in a form that applies to our offline non-uniform revenue setting. This modified SBP algorithm starts by partitioning the total time limit into *time windows*, where each window (except possibly the last) is split into two equal *time segments*. The algorithm uses the first segment of each window to determine a maximum revenue set of requests that can be served within a segment, moving (if needed) to this set. It then uses the second segment of each window to serve the requests in this set.

For a literature review of some of the numerous DARP variants, see a recent survey by Ho et al. [8]. DARP problems are generalizations of the Traveling Salesperson Problem (TSP), so we mention TSP work that is most closely related to the time-limited variant of DARP that we study in this paper. Balas [2] first introduced the Prize Collecting Traveling Salesperson Problem (PCTSP), in which the server earns a prize (similar to our revenues) for each location visited, with the goal of collecting a prescribed amount of prize money

24th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2024).
Editors: Paul C. Bouman and Spyros C. Kontogiannis; Article No. 8; pp. 8:1–8:6
OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

while minimizing travel costs and penalties. Bienstock et al. [3] gave the first approximation algorithm for PCTSP with ratio 2.5. Recently, Blauth and Nägele [4] achieved a significant improvement, obtaining an approximation guarantee of 1.774. Blum et al. [5] provide a constant-factor approximation algorithm for the Orienteering Problem (OP), another special case of the DARP problem we study. The goal of OP is, given a weighted graph with rewards on the nodes, to maximize the total reward collected on a path of a predefined maximum length. Our problem generalizes OP (and TSP), since in DARP we must visit pairs of points, rather than single points. The limit on the path length for OP is analogous to the time limit T in our DARP setting.

In this work, we highlight how SBP's performance changes when the revenues switch from uniform to non-uniform. Previously, we showed that an adapted version of the SBP algorithm which enforced an even number of time segments gave an approximation ratio of 4 in the uniform-revenue setting [1]. In this work we show that when revenues are non-uniform, SBP approximates the optimal revenue that can be earned within the time limit to within a factor of 5. We then show that when the number of time segments is odd, the approximation ratio of SBP is no better than 5, before showing that when the number of time segments is even, the ratio is strictly greater than 4.

2 New Upper and Lower Bounds

We formally define RDARP, the *Revenue-Dial-a-Ride-Problem*, as follows. The input to RDARP is a complete weighted graph, a set of requests given as source-destination node-pairs where each request has an associated revenue, and a time limit $T > 0$. We note that any simple, connected, weighted graph is allowed as input, with the simple preprocessing step of adding an edge wherever one is not present whose weight is the length of the shortest path between its two endpoints. We further note that the input can be regarded as a metric space if the graph is undirected and the edge weights satisfy triangle inequality. We treat the edge weights as travel-times, but for expository convenience may also refer to them as distances. Let t_{max} denote the maximum length of an edge in the graph.

Algorithm 1 SEGMENTED BEST PATH (SBP) Algorithm as adapted from [6]. Input: time limit $T > 0$, a complete graph with $T \geq 2t_{max}$, and a set of requests with associated revenues.

```
Let $t_1, t_2, \dots t_f$ denote time segments of length $X = T/f$ ending at times
  $T/f, 2T/f, \dots, T$, respectively, where $f = \lfloor T/t_{max} \rfloor$.
Let $i = 1$.
while $i < f$ and there are still unserved requests do
    At the start of $t_i$, find the max-revenue-sequence, $\mathcal{R}$.
    Move to the source location of the first request in $\mathcal{R}$.
    At the start of $t_{i+1}$, serve the requests in $\mathcal{R}$.
    Let $i = i + 2$.
end while
```

We begin by adapting the SBP algorithm for online DARP to the offline RDARP setting. In the online setting, SBP was shown in [6] to have competitive ratio 6, which was then improved to 5 and shown to be tight [7]. At the beginning of SBP (see Algorithm 1), set f, the number of time segments, to $\lfloor T/t_{max} \rfloor$. Let $X = T/f$ be the length of a *time segment*; note $X \geq t_{max}$. Let every pair of consecutive time segments, starting from the first time segment t_1, form a *time window*. A *max-revenue-sequence*, $\mathcal{R}$, is a sequence of requests of maximum total revenue that can be served within one time segment of length T/f.

We use the term *drive* to refer to any move of the server from one point to another, whether or not there is a request being served during the move. We use OPT to denote an optimal solution: a sequence of requests that maximizes total revenue that can be served within the time limit T. We let $|\textsc{opt}(I)|$ denote the total revenue earned by the optimal solution on an instance I of RDARP.

2.1 Upper bound

In what follows, we allow OPT to choose its desired position at time 0. We will show that even with this extra flexibility, SBP is a 5-approximation.

▶ **Lemma 1.** *Let r denote the revenue of all requests that* OPT begins *serving by the end of the first time window, and s denote the revenue earned by* SBP *within the first time window. Then $s \geq r/4$.*

Proof. Consider the initial subpath of OPT that contains the first time window and any requests OPT begins serving by the end of the first time window. This subpath thus has revenue r. We subdivide this subpath into four further subpaths:

1. The subpath that is entirely contained in the time interval $[0, X]$.
2. The subpath that is entirely contained in the time interval $[X, 2X]$.
3. The drive (not necessarily a request), if any, between (1) and (2).
4. The drive (not necessarily a request), if any, that comes after (2) and overlaps time $2X$.

Each of these four subpaths has total length no greater than X. Because their collective revenue is r, at least one of the four subpaths must have revenue at least $r/4$. Since SBP could have greedily chosen any of these four subpaths, SBP must thus earn revenue $s \geq r/4$. ◀

▶ **Theorem 2.** *For the offline general metric with nonuniform revenues and time limit $T \geq 2t_{max}$, we have $|\textsc{opt}| \leq 5|SBP|$.*

Proof. We will show by induction on the number of time windows that $|\textsc{sbp}| \geq |\textsc{opt}|/5$.

Base case 1: There is only one time window, so $T = 2X$. ($T \geq 2X$ since $T \geq 2t_{max}$ by assumption, so $f \geq 2$.) Using Lemma 1, $|\textsc{opt}| = r \leq 4s = 4|\textsc{sbp}| \leq 5|\textsc{sbp}|$.

Base case 2: $T = 3X$. For this case, consider the above proof of Lemma 1, but add one more subdivision so that there are 5 subpaths instead of 4. The fifth subpath is the subpath of OPT that is entirely contained within the time interval $[2X, 3X]$. We now redefine r to be the revenue earned by OPT by time $3X$. The rest of the proof remains the same, except we divide into five subpaths instead of four, so $|\textsc{opt}| = r \leq 5s = 5|\textsc{sbp}|$.

For the inductive step, we may now assume $T \geq 4X$, with the theorem holding for any smaller time limit than T. Let I refer to the original input instance. We want to show that $|\textsc{sbp}(I)| \geq |\textsc{opt}(I)|/5$. We consider the smaller instance after SBP has completed its first time window. This smaller instance, I', has time limit $T - 2X$, and the requests served by SBP in the first time window are removed.

Consider the path in I' formed by taking the OPT path in I and removing the initial part that earned revenue r; this path has length at most $T - 2X$, and revenue at least $|\textsc{opt}(I)| - r - s$. In essence, it is the OPT path with the first portion removed and potentially some 'holes' from requests that are not present in I'. By the inductive hypothesis, $\textsc{sbp}(I')$ would have revenue at least $|\textsc{opt}(I')|/5 \geq (|\textsc{opt}(I)| - r - s)/5$.

Thus, using the inductive hypothesis and Lemma 1, we have $|\textsc{sbp}(I)| = s + |\textsc{sbp}(I')| \geq s + (|\textsc{opt}(I)| - r - s)/5 = (|\textsc{opt}(I)| - r + 4s)/5 \geq (|\textsc{opt}(I)| - r + r)/5 = |\textsc{opt}(I)|/5$, completing the induction. ◀

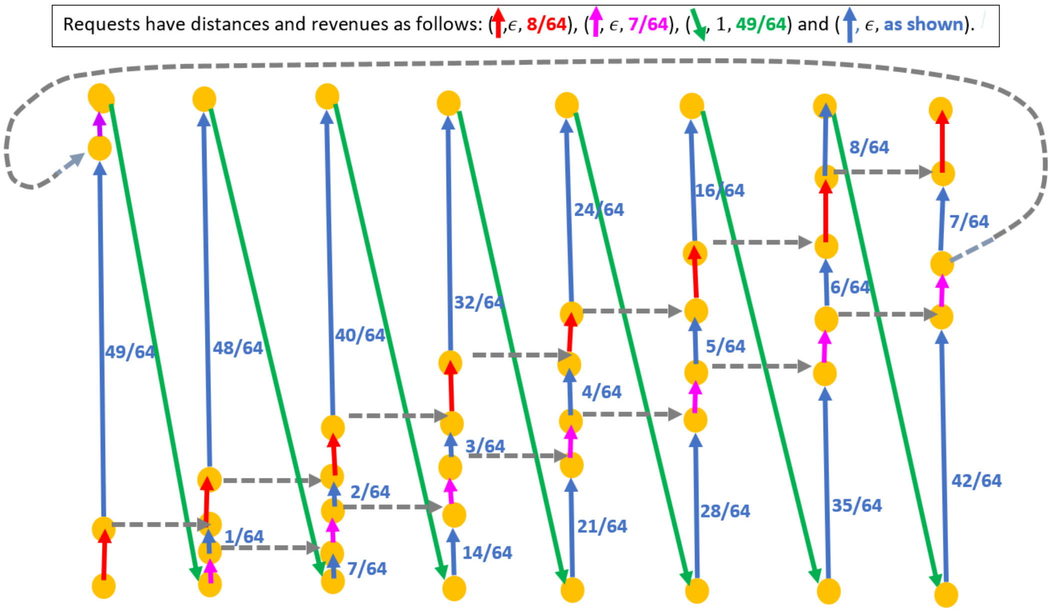

Requests have distances and revenues as follows: (↑, ϵ, 8/64), (↑, ϵ, 7/64), (↘, 1, 49/64) and (↑, ϵ, as shown).
49/64
48/64
40/64
32/64
24/64
16/64
8/64
7/64
6/64
5/64
4/64
3/64
2/64
1/64
7/64
14/64
21/64
28/64
35/64
42/64

Figure 1 A small instance with $a = 4$ and $b = 2$ that illustrates the overall structure of the lower bound (but does not yield a ratio greater than 4). Requests are shown in color with distances and revenues as indicated. Gray dashed edges are empty drives of distance ϵ. All edges not shown have distance 1, including the reverse of existing directed edges.

2.2 Tight lower bound when the number of time segments is odd

Consider an instance with $T = 6$, $t_{max} = 2$, and five (or more) requests of unit revenue and length $1 + \delta$, for some small $\delta > 0$, so $f = \lfloor T/t_{max} \rfloor = 3$. These requests are such that OPT is able to serve five consecutively until the time limit, earning total revenue of 5. By contrast, SBP serves only one request: two consecutive requests take time $2 + 2\delta > 2 = T/f$, so only one request can be served within a single time segment of length T/f, and the first segment of each window is used, by definition of SBP, to move. Thus, we achieve a ratio of 5 which matches base case 2 of Theorem 2.

In [1] we proposed a version of SBP that ensured f, the number of time segments, was even. There we showed that version of SBP earned a (tight) approximation ratio of 4; however, in what follows we show that even if the number of time segments is even in Algorithm 1, we cannot guarantee an upper bound of 4 on the ratio.

2.3 Lower bound when the number of time segments is even

We describe how to construct instances with an even number of time segments that have a lower bound greater than 4. We note that these instances are not metric spaces, as they lack symmetry. Let a, b be positive integers with $b \leq a$. Let $t_{max} = 1$ and $T = 2a$, resulting in $f = 2a$ and $X = 1$. See Figure 1 for a representative example with $a = 4$ and $b = 2$.

Let the OPT path consist of $P_1, E_1, P_2, E_2, ..., P_{2a-1}, E_{2a-1}, P_{2a}$ where each P_i (depicted in Fig. 1 as vertical paths oriented upward) is a path of up to $2b + 1$ requests with total distance less than $1/(2a)$ and total revenue 1, and each E_i (depicted as a green downward diagonal edge in the figure) is a request of distance 1 and revenue c that we will specify later. These are the only requests in the input. Hence, the OPT path has total distance at most $2a/(2a) + 2a - 1 = 2a$, and the optimal solution can be completed in time T.

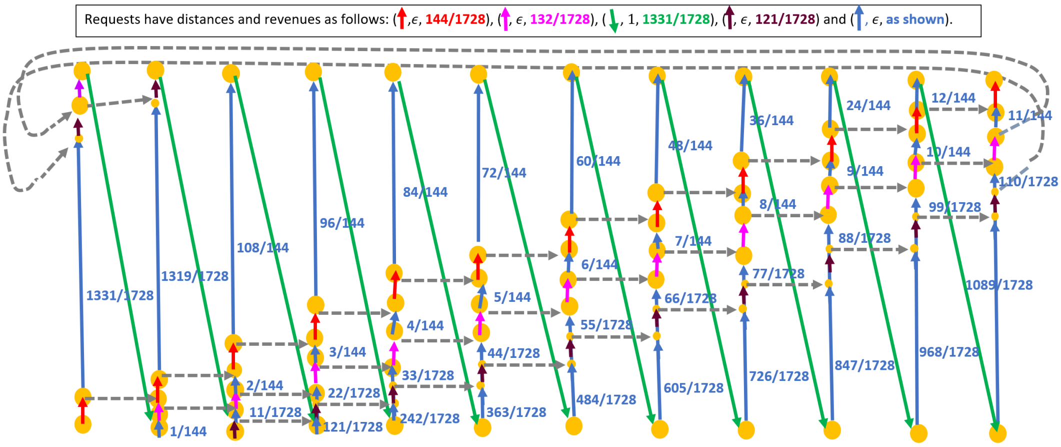


Figure 2 Instance with $a = 6, b = 3$. Requests are shown in color with distances and revenues as indicated. Gray dashed edges are empty drives of distance ϵ. All edges not shown have distance 1, including the reverse of existing directed edges.

All other distances between nodes will be 1 except that specific nodes of each P_i are very close to nodes in P_{i+1} so that there are paths $Q_1, Q_2, \ldots, Q_b$ defined as follows. Q_1 (depicted in Figures 1 and 2 as the red and gray staircase pattern) starts with a request in P_1 of revenue $1/(2a)$ and then cuts through $P_2, P_3, \ldots P_{2a}$, serving a request of revenue $1/(2a)$ from each P_i, followed by an empty drive (gray dashed edges) of a sufficiently small distance $\epsilon > 0$ after each request, accruing a total revenue of 1. (Note: ϵ must be small enough so that Q_1 can be served within one time segment.)

Setting $c \leq 1$, no paths of total distance 1 or less have revenue larger than 1, so we can assume that SBP will move to and then drive along Q_1 during the first two time segments. At time $t = 2$, each path P_i now has remaining revenue $(1 - \frac{1}{2a})$, which we denote by ρ.

Now suppose we have path Q_2 (depicted as the magenta and gray staircase in the figures) that similarly cuts through $P_2, P_3, \ldots, P_{2a}, P_1$, serving a request of revenue $\rho/(2a)$ each, so that Q_2 has total revenue ρ. Again, as long as $c \leq \rho$, no path of length 1 or less has revenue more than 1 at time $t = 0$, and no path of length 1 or less has revenue more than ρ at time $t = 2$. So we can assume that SBP moves to and serves Q_2 from time $t = 2$ to time $t = 4$.

In general, define Q_i for $i = 1 \ldots b$ as a path that cuts through $P_i, P_{i+1} \ldots P_{2a}, P_1 \ldots P_{i-1}$, serving a request of revenue $\rho^{i-1}/(2a)$ each, so that Q_i has a total revenue of ρ^{i-1}. To ensure that SBP chooses Q_i in time segment $[2i - 2, 2i]$, we need $c \leq \rho^{i-1}$, for all $i = 1 \ldots b$. Note that each Q_i, $1 \leq i \leq b$, consists of $4a - 1$ drives: $2a$ requests with an empty drive of distance ϵ between each pair for a total of $2a - 1$ empty drives.

After time $t = 2b$, the remnants of each path P_i have revenue ρ^b. (After SBP serves Q_i for $i = 1 \ldots b$, the remaining revenue of each P_i shrinks by a factor of ρ.) If $c \geq \rho^b$, SBP may serve paths of revenue c for the remainder of time. We choose $c = \rho^b$, so SBP serves $a - b$ paths of revenue ρ^b for the remainder of time. Summarizing,

$$|\text{OPT}| = 2a + (2a-1)\rho^b = 2a + (2a-1)\left(1 - \frac{1}{2a}\right)^b$$

and since $1 - \rho = 1/(2a)$,

$$\begin{aligned}|\text{SBP}| &= 1 + \rho + \rho^2 + \ldots \rho^{b-1} + (a-b)\rho^b = (1-\rho^b)/(1-\rho) + (a-b)\rho^b \\ &= 2a(1-\rho^b) + (a-b)\rho^b = 2a - \rho^b(a+b) = 2a - \left(1 - \frac{1}{2a}\right)^b (a+b).\end{aligned}$$

Table 1 Some sample instance parameters and their corresponding ratios.

b	a	$\lvert\text{OPT}\rvert/\lvert\text{SBP}\rvert$
2	6	4.025
3	6	4.03985
3	10	4.09867
1000000	1877946	4.27805

If $a = b$, we can take the limit as $a \to \infty$ to get $\rho^b = (1 - \frac{1}{2a})^a$ is $1/\sqrt{e}$. Then $\lvert\text{OPT}\rvert/\lvert\text{SBP}\rvert$ has a limit of $(\sqrt{e}+1)/(\sqrt{e}-1) \approx 4.08299$.

Table 1 shows some sample instance parameters and their corresponding ratios. The instance shown in Figure 2 is reflected in the second row. Thus far, preliminary testing suggests that a ratio much greater than 4.27805 (in the final row of the table) is unachievable.

Since our upper bound is tight only when f is odd, we continue to investigate a version of the SBP algorithm that enforces an even number of time segments. An open question is if the upper bound of 5 is no longer tight for this adjusted algorithm, and whether the true upper bound matches the above family of instances, or can be shown to be strictly below 5.

References

1 Barbara M. Anthony, Ananya D. Christman, Christine Chung, and David Yuen. Serving Rides of Equal Importance for Time-Limited Dial-a-Ride. In Panos Pardalos, Michael Khachay, and Alexander Kazakov, editors, *Mathematical Optimization Theory and Operations Research*, pages 35–50. Springer International Publishing, 2021. doi:10.1007/978-3-030-77876-7_3.

2 Egon Balas. The prize collecting traveling salesman problem. *Networks*, 19(6):621–636, 1989. doi:10.1002/net.3230190602.

3 Daniel Bienstock, Michel X. Goemans, David Simchi-Levi, and David Williamson. A note on the prize collecting traveling salesman problem. *Mathematical programming*, 59(1-3):413–420, 1993. doi:10.1007/BF01581256.

4 Jannis Blauth and Martin Nägele. An improved approximation guarantee for prize-collecting TSP. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, STOC 2023, pages 1848–1861, New York, NY, USA, 2023. doi:10.1145/3564246.3585159.

5 Avrim Blum, Shuchi Chawla, David R. Karger, Terran Lane, Adam Meyerson, and Maria Minkoff. Approximation algorithms for orienteering and discounted-reward TSP. *SIAM Journal on Computing*, 37(2):653–670, 2007. doi:10.1137/050645464.

6 Ananya Christman, Christine Chung, Nicholas Jaczko, Marina Milan, Anna Vasilchenko, and Scott Westvold. Revenue Maximization in Online Dial-A-Ride. In *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017)*, volume 59, pages 1:1–1:15, Dagstuhl, Germany, 2017. doi:10.4230/OASIcs.ATMOS.2017.1.

7 Ananya D. Christman, Christine Chung, Nicholas Jaczko, Tianzhi Li, Scott Westvold, Xinyue Xu, and David Yuen. Improved Bounds for Revenue Maximization in Time-Limited Online Dial-a-Ride. *SN Operations Research Forum*, 2(3):1–38, September 2021. doi:10.1007/S43069-021-00076-X.

8 Sin C. Ho, W.Y. Szeto, Yong-Hong Kuo, Janny M.Y. Leung, Matthew Petering, and Terence W.H. Tou. A survey of dial-a-ride problems: Literature review and recent developments. *Transportation Research Part B: Methodological*, 111:395–421, 2018.

Online Vehicle Routing with Pickups and Deliveries Under Time-Dependent Travel-Time Constraints

Spyros Kontogiannis

Computer Engineering and Informatics Department, University of Patras, Greece
Computer Technology Institute and Press "Diophantus", Patras, Greece

Andreas Paraskevopoulos

Computer Technology Institute and Press "Diophantus", Greece

Christos Zaroliagis

Computer Engineering and Informatics Department, University of Patras, Greece
Computer Technology Institute and Press "Diophantus", Patras, Greece

Abstract

The *Vehicle Routing Problem with pickups, deliveries and spatiotemporal service constraints* ($\mathbb{VRP}_{\text{PDSTC}}$) is a quite challenging algorithmic problem that can be dealt with in either an off-line or an online fashion. In this work, we focus on a generalization, called $\mathbb{VRP}_{\text{PDSTCtd}}$, in which the travel-time metric is *time-dependent*: the traversal-time per road segment (represented as a directed arc) is determined by some function of the departure-time from its tail towards its head. Time-dependence makes things much more complicated, even for the simpler problem of computing earliest-arrival-time paths which is a crucial subroutine to be solved (numerous times) by $\mathbb{VRP}_{\text{PDSTCtd}}$ schedulers. We propose two *online* schedulers of requests to workers, one which is a time-dependent variant of the classical `Plain-Insertion` heuristic, and an extension of it trying to digest some sort of forecasts for future demands for service. We enrich these two online schedulers with two additional heuristics, one targeting for distance-balanced assignments of work loads to the workers and another that makes local-search-improvements to the produced solutions. We conduct a careful experimental evaluation of the proposed algorithms on a real-world instance, with or without these heuristics, and compare their quality with human-curated assignments provided by professional experts (human operators at actual pickup-and-delivery control centers), and also with feasible solutions constructed from a relaxed MILP formulation of $\mathbb{VRP}_{\text{PDSTCtd}}$, which is also introduced in this paper. Our findings are quite encouraging, demonstrating that the proposed algorithms produce solutions which (i) are significant improvements over the human-curated assignments, and (ii) have overall quality pretty close to that of the (extremely time-consuming) solutions provided by an exact solver for the MILP formulation.

2012 ACM Subject Classification Theory of computation → Theory and algorithms for application domains

Keywords and phrases transport optimization heuristics, vehicle routing with pickups and deliveries, time-dependent travel-times

Digital Object Identifier 10.4230/OASIcs.ATMOS.2024.9

Related Version *Extended Version*: `https://doi.org/10.48550/arXiv.2408.06324` [10]

Funding This work was co-financed by the European Regional Development Fund of EU and Greek national funds through the Operational Program Competitiveness, Entrepreneurship and Innovation (call RESEARCH-CREATE-INNOVATE) under contract no. T2EDK-03472 (project "i-Deliver").

1 Introduction

The vehicle routing problem with pickups, deliveries and spatiotemporal service constraints, $\mathbb{VRP}_{\text{PDSTC}}$, concerns the utilization of a fleet of *workers* (e.g., drivers, couriers, etc.) with their own work-shifts and capacitated vehicles, for the provision of one-to-one delivery services of

24th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2024).
Editors: Paul C. Bouman and Spyros C. Kontogiannis; Article No. 9; pp. 9:1–9:20
OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

commodities (e.g., parcels, food, individuals, etc.) from their origins (*pickup points*) to their destinations (*delivery points*) within certain hard time-windows which are determined by earliest pickup-times and latest delivery-times per commodity. The primary goal is to have a maximum number of served commodity-delivery requests by the fleet of workers, respecting all spatiotemporal constraints (i.e., vehicle capacities, work shifts, and servicing time-windows), with a secondary objective that the workers commute in an underlying road network of a (typically large-scale) urban area in such a way that a specific aggregate service-cost function (e.g., total travel-time, total-distance of the entire fleet, etc.) is minimized.

An even more complicated generalization of the problem, $\mathbb{VRP}_{\mathrm{PDSTCtd}}$, considers instances in which the traversal-times of the road segments (which are represented as directed arcs), rather than being scalars, are *time-dependent*, i.e., they are determined by given arc-traversal-time functions of the departure-times from their tails towards their heads. Such a travel-time metric is typical when computing earliest-arrival-time paths for private vehicles commuting within road networks, but unfortunately makes the problem of computing earliest-arrival-time paths much harder (cf. [11] and references therein). Since this is a typical subroutine that must be used numerous times when solving an instance of $\mathbb{VRP}_{\mathrm{PDSTCtd}}$, it is clear that this generalization of the vehicle routing problem becomes even harder as well.

The problem can be dealt with either offline, i.e., having at the solver's disposal the entire instance of delivery requests to be served and the fleet of workers, or online, i.e., when the requests for commodities to be delivered appear in real-time and/or the workers are activated at will. For $\mathbb{VRP}_{\mathrm{PDSTC}}$, typical approaches for the offline case such as the consideration of an appropriate mixed-integer linear programming formulation and the use of state-of-the-art MILP solvers, are well-known but also extremely demanding in computational resources, since the problem is NP-hard to solve. Unfortunately, for $\mathbb{VRP}_{\mathrm{PDSTCtd}}$ the situation becomes even more complicated, since there is no MILP formulation to solve (the travel-time metric is not constant but time-dependent).

Therefore, our focus is mainly on the efficient construction of suboptimal solutions in an online scenario where the work-shifts are predetermined and a priori known to the scheduler, but the requests are revealed in real-time and the scheduler has to always maintain a feasible solution for a maximal number of the active requests by the currently operational workers. As it is not obvious how classical constructive and improvement heuristics for $\mathbb{VRP}$ can be adapted when the service requests come in pickup-delivery pairs (one per served commodity), the literature for $\mathbb{VRP}_{\mathrm{PDSTC}}$ has mainly focused on simple online solvers, namely some well-known constructive heuristics such as the `Neighborhood` and the `Insertion` heuristics. In particular, `Insertion` is a popular online algorithm, heavily used and experimented in the past, e.g., in [3, 16] for $\mathbb{VRP}_{\mathrm{PDSTC}}$, which simply constructs incrementally a feasible solution by allocating in a locally-optimal way each emergent request to one of the existing routes (creating a new route also being an option, provided there exist active workers still awaiting their first assignment) in such a way that the relative order of the already assigned requests remains intact and the incremental cost in the value of the objective function is minimized. Typically this heuristic requires cubic time, but there are also some quite efficient (even linear-time) implementations based on dynamic programming [16]. Incorporating a time-dependent travel-time metric in such heuristics is already a challenge. On the other hand, the adaptation of well-known exact polynomial-size MILP formulations for $\mathbb{VRP}_{\mathrm{PDSTC}}$ to $\mathbb{VRP}_{\mathrm{PDSTCtd}}$, to be fed to an offline solver, seems to be very hard because the point-to-point travel-times are now time-dependent variables rather than scalars.

In this work, we propose, implement, engineer, and experimentally evaluate two insertion-based algorithms for $\mathbb{VRP}_{\mathrm{PDSTCtd}}$: The `TD-Insertion` and the `TD-Prophet`. Our implementation of `TD-Insertion`, apart from the typical greedy criterion (the minimization of the additive cost for fitting a new request in the subtour of a worker) for accommodating an emergent commodity to some active worker, also considers an alternative local-optimization criterion, which essentially attempts to keep a rough balance in the aggregate lengths of the workers' subtours. This criterion was inspired by [1] who observed, in the most elementary variant of $\mathbb{VRP}_{\mathrm{C}}$ only with vehicle capacities, that in optimal solutions some subtours correspond to much longer routes than others. Trying to avoid this kind of unfairness among the workers' actual commodity-servicing tasks, they proposed to compute the scores (i.e., marginal increases in route lengths) of the candidate pairs using the *difference of squared costs* (the ℓ_2-scoring criterion), rather than just the difference of the route costs (the ℓ_1-scoring criterion). We implement and experimentally evaluate for $\mathbb{VRP}_{\mathrm{PDSTCtd}}$ the local-optimization analogue of the ℓ_2-scoring criterion for our `TD-Insertion` heuristic. As an alternative, we also try to hard-code fairness in the workers' subtour lengths when considering the classical ℓ_1-scoring criterion, via an additional heuristic feature that we may opt to use in our scheduler, called the `Work Balancer` heuristic.

Our `TD-Prophet` algorithm was inspired by the `Prophet-Insertion` algorithm of [16] and works similarly with `TD-Insertion`, but also tries to account for some sort of *forecasts* for near-future requests and handles them exactly as the actual requests. Apart from the consideration of the time-dependent travel-time metric, another difference from the `Prophet-Insertion` algorithm of [16] is that `TD-Prophet` does not have the workers always on the move just because of predictions for the entire period (as `Prophet-Insertion` does in the static case); it just fits a small number of *short-term* predictions (e.g., only within the next hour of operation) to their actual assignment of real requests that appear online to the system, and simply shortcuts the coverage of delivery points of those predictions that were not eventually verified in real-time at their pickup points.

Apart from implementing and engineering our online algorithms for $\mathbb{VRP}_{\mathrm{PDSTCtd}}$, we also evaluate the efficiency of a local-search improvement heuristic, namely, the repetitive `Relocation` of already assigned but not yet served routes right after handling a new (real or predicted) request, towards improving the solutions produced by the two algorithms.

As it would be too expensive to have an *exact* mixed-integer linear programming (MILP) formulation for $\mathbb{VRP}_{\mathrm{PDSTCtd}}$[1], we also propose a heuristic construction of some "baseline" solutions, using a *relaxed* MILP formulation of polynomial size. This MILP considers a *scalar* travel-time metric for interconnecting routes of service points of the requests which, rather than being just the average travel-times or the (optimistic) free-flow travel-times or the (pessimistic) full-congestion metric, are deduced by some "educated" estimations (scalars) of the actual time-dependent travel-times, depending on when these interconnecting routes are most likely to be used by any worker. Well-known MILP solvers are then used to create, within bounded execution time, a small set of solutions which are then tested for feasibility w.r.t. the temporal constraints, under the actual (time-dependent) travel-time metric. This way we get some "baseline solutions" with which the solutions of our online algorithms are compared. Of course, even an optimal solution for the relaxed MILP is not necessarily an optimal solution for the time-dependent instance at hand, and its cost does not necessarily constitute some guaranteed lower-bound of the optimal cost. Of course, this MILP-based method is rather unrealistic for our online scenario, due to both the assumption of a priori knowing all delivery requests and its need for extremely demanding computational resources.

[1] One could possibly consider a set-partitioning formulation using all feasible routes, but this would require exponentially many variables.

Finally, we conduct a thorough experimental evaluation of our online algorithms for $\mathbb{VRP}_{\text{PDSTCtd}}$, with or without the heuristic improvements, on a real-world instance of food and supermarket delivery requests in an urban environment, which is fed with synthetic demand forecasts of varying accuracy. It is mentioned at this point that although high-quality demand forecasting is of paramount importance, it is *not* the subject of the present work. This is why, for the purposes of our experimental evaluation only, we created synthetic forecasting data of varying accuracy. As a measure of comparison for our produced solutions, we use both the feasible solutions constructed by the relaxed MILP formulation and the actual solution that was determined by experienced human operators in our real-world dataset. Our results demonstrate a significant prevalence of both our online algorithms over the human-curated solution, up to 49% in total length and travel time, and also the prevalence of `TD-Prophet` over `TD-Insertion`, up to 4%, on finding better pickup-delivery scheduling solutions.

2 Problem Statement and Related Work

We are given a sequence $\mathcal{R} = \langle r_1, r_2, \ldots, r_{|\mathcal{R}|}\rangle$ of *pickup-and-delivery* requests. Each request is a tuple $r = \left(\chi_r^{pic}, t_r^{ep}, t_r^{psrv}, \chi_r^{del}, t_r^{ld}, t_r^{dsrv}, q_r, h_r\right) \in \mathcal{R}$, where: t_r^{ep} (t_r^{ld}) is the *earliest-pickup-time* (*latest-delivery-time*) that a worker may receive (leave) the commodity from (at) the *pickup point* χ_r^{pic} (*delivery point* χ_r^{del}), assuming that $t_r^{ep} < t_r^{ld}$; t_r^{psrv} (t_r^{dsrv}) is the anticipated *service-time* for the worker that is responsible for commodity r, at the corresponding (pickup or delivery) location (χ_r^{pic} or χ_r^{del}); q_r is the *volume/weight* of the commodity to be transferred, that is consumed from the corresponding vehicle's capacity; and $h_r \subseteq \mathcal{H}$ is the subset of *eligible vehicle-types* for the good to be transferred (e.g., bicycle, motorcycle, car, with a cooler or heated box, etc.). $\mathcal{V}^{pic} = \{(r, \chi_r^{pic}) : r \in \mathcal{R}\}$ and $\mathcal{V}^{del} = \{(r, \chi_r^{del}) : r \in \mathcal{R}\}$ are the sets of pickup and delivery events, respectively, for all the active requests in $\mathcal{R}$. It is noted that, even if two requests $r \neq r'$ share some (geographical) service point, e.g., $\chi_r^{pic} = \chi_{r'}^{del}$ or $\chi_r^{pic} = \chi_{r'}^{pic}$, the pairs $(r, \chi_r^{pic}), (r, \chi_r^{del}), (r', \chi_{r'}^{pic}), (r', \chi_{r'}^{del})$ are distinct.

There is also a set $\mathcal{W} = \left\{w_1, w_2, \ldots, w_{|\mathcal{W}|}\right\}$ of active workers (e.g., operational couriers during a work-shift), each of them represented by a tuple $w = (\chi_w^{start}, t_w^{start}, \chi_w^{end}, t_w^{end}, Q_w, H_w)$ where: χ_w^{start} and t_w^{start} (χ_w^{end} and t_w^{end}) are the initial (final) *location* and opening (closing) time, respectively, of w's work-shift; $H_w \in \mathcal{H}$ is the *type* of the particular vehicle used by w (e.g., bicycle, motorcycle, car, etc); Q_w is the maximum *volume/weight* of storage, for the vehicle used by w. $\mathcal{V}^{start} = \{(w, \chi_w^{start}) : w \in \mathcal{W}\}$ and $\mathcal{V}^{end} = \{(w, \chi_w^{end}) : w \in \mathcal{W}\}$ are the sets of work-shift starting and finishing locations, for all the active workers.

Each worker $w \in \mathcal{W}$ may be assigned an arbitrary subset of requests $\mathcal{R}_w \subseteq \mathcal{R}$ which are eligible for them to serve. The whole task for w is represented as a sequence of all the corresponding pickup and delivery points for requests of $\mathcal{R}_w$, called his/her *subtour*. Then, w is assumed to move within an urban area along earliest-arrival-time subpaths connecting consecutive points in the subtour, in order to serve them. The area is represented by a directed graph $G = (V, E)$, whose arcs correspond to unidirectional road segments and vertices represent intersections and intermediate points (corresponding to distinct postal addresses) of these road segments. Each arc $e = uv \in E$ comes with a scalar *arc-length*, $\lambda[e]$, and a periodic *arc-travel-time* function $\tau_h[e](t) : [0, T) \mapsto \mathbb{R}_{\geq 0}$ for evaluating the traversal-time of e when using a vehicle of type $h \in \mathcal{H}$, depending on the departure-time from u. For succinctness in its representation, this function is assumed to be continuous and piecewise linear (pwl), represented as a constant-size sequence of breakpoints. It is also assumed to satisfy the FIFO property, as is typical for individually moving private vehicles within road networks. The FIFO property implies that the corresponding *arc-arrival-time* function

$a_h[e](t) = t + \tau_h[e](t)$ for e when using h is *non-decreasing*. In a similar fashion, we inductively define the notions of travel-time and arrival-time functions for paths which are perceived as sequences of incident arcs: For each $k \geq 0$, a path $\pi = \langle e_1 = (i_0, i_1), \ldots, e_k = (i_{k-1}, i_k) \rangle$ and an arc $e_{k+1} = (i_k, i_{k+1})$, the path $\pi \oplus e_{k+1}$ is constructed by appending e_{k+1} at the end of π. It then holds that $\lambda[\pi \oplus e_{k+1}] = \lambda[\pi] + \lambda[e_{k+1}]$, $a_h[\pi \oplus e_{k+1}](t) = a_h[\pi](t) + \tau_h[e_{k+1}](a_h[\pi](t))$, and $\tau_h[\pi \oplus e_{k+1}](t) = a_h[\pi \oplus e_{k+1}](t) - t$. Furthermore, $\tau_h[o,d](t_o)$ denotes the minimum path-travel-time, when departing at time t_o from $o \in V$ towards $d \in V$, using a vehicle of type h, and the earliest arrival-time at d is denoted as $a_h[o,d](t) = \tau_h[o,d](t) + t$. The scalar $\lambda[o,d]$ denotes the minimum path-length from o to d.

A *feasible solution* for an instance of $\mathbb{VRP}_{\text{PDSTCtd}}$ is described as a collection $\{S_w : w \in \mathcal{W}\}$ of *subtours* (i.e., sequences of service points for all the requests assigned to them), one per worker, such that each request belongs to at most one subtour and, along each subtour S_w, there is no violation of a temporal constraint or a vehicle capacity constraint as w moves with his/her vehicle between consecutive service points along S_w across interconnecting paths of the road graph G. The primary goal is to find a feasible solution that maximizes the number of served (i.e., assigned) requests, and a secondary goal is to minimize a global cost objective value (e.g., total travel-time or total-length, for all workers).

For convenience, we consider a special graph, the *pickup-and-delivery* (PD) graph $G_{PD} = (\mathcal{V}, \mathcal{E})$ (cf. Figure 1), whose node set $\mathcal{V}$ contains four subsets of nodes corresponding to distinct events: The green and orange nodes correspond to workers-shift starting and ending events from $\mathcal{V}^{start}$ and $\mathcal{V}^{end}$, respectively. The purple and blue nodes correspond to pickup and delivery events from $\mathcal{V}^{pic}$ and $\mathcal{V}^{del}$, respectively. As for the arc set $\mathcal{E}$, nodes from $\mathcal{V}^{start}$ are connected to all nodes in $\mathcal{V}^{pic}$, nodes from $\mathcal{V}^{del}$ are connected to each node in $\mathcal{V}^{end}$, and (roughly) a complete subgraph is induced by $\mathcal{V}^{pic} \cup \mathcal{V}^{del}$, excluding only arcs from each delivery event to the pickup event of the same request, as they cannot be part of any solution.

For each $u \in \mathcal{V}^{pic} \cup \mathcal{V}^{del}$, $\rho(u) \in \mathcal{R}$ denotes the corresponding request. For each $v \in \mathcal{V}^{start} \cup \mathcal{V}^{end}$, $\gamma(v) \in \mathcal{W}$ denotes the corresponding worker for the work-shift v. Within G_{PD}, each subtour $S_w = \langle v_0, v_1, \ldots, v_{k+1} \rangle$ can be seen as a (simple) path where $v_0 = \chi_w^{start}$, $v_{k+1} = \chi_w^{end}$, and $\forall i \in \{1, 2 \ldots, k\}, v_i \in \mathcal{V}^{pic} \cup \mathcal{V}^{del}$ (service point for some request).

For each arc $uv \in \mathcal{E}$, there is a minimum-length path $\pi_{u,v}^{\lambda}$ (and possibly suboptimal travel-time), and a minimum-travel-time path $\pi_{u,v}^{\tau}(t_u)$, dependent on the departure-time t_u (and possibly suboptimal length) in the underlying road graph G connecting u and v. Each subtour (i.e., simple path in G_{PD}) S_w of a worker can then be translated within the road graph G into a route by using either a distance-optimizing *route* Π_w^{λ} (prioritizing the usage of length-optimal interconnecting paths), or a travel-time-optimizing route $\Pi_w^{\tau}(t_w^{start})$ (prioritizing the usage of length-optimal interconnecting paths), that interconnects all the consecutive points in S_w. For some route $\Pi_w = \left(v_0 = \chi_w^{start}, v_1, \ldots, v_{k+1} = \chi_w^{end}\right)$ for worker $w \in \mathcal{W}$, we denote by $\Pi_w^{u:v}$ the subroute starting at node u and ending at node v. For a given departure-time t, we associate each node $v_i \in \Pi_w$ with the following labels:

1. *arrival-time*: $a(v_i) = a[\Pi_w^{v_0:v_i}](t)$;
2. *earliest departure-time*: $d(v_i) = \begin{cases} \max\{a(v_i), t_{\rho(v_i)}^{ep}\} + t_{\rho(v_i)}^{psrv}, & v_i \in \mathcal{V}^{pic} \\ a(v_i) + t_{\rho(v_i)}^{dsrv}, & v_i \in \mathcal{V}^{del} \\ a(v_i), & \text{otherwise} \end{cases}$
3. *waiting-time*: $b(v_i) = \max\{t_{\rho(v_i)}^{ep} - a(v_i), 0\}$ for $v_i \in \mathcal{V}^{pic}$, and $b(v_i) = 0$ otherwise; and
4. *current-load*: $C(v_0) = 0;\ \forall i \geq 1, C(v_i) = \begin{cases} C(v_{i-1}) + q_{\rho(v_i)}, & v_i \in \mathcal{V}^{pic} \\ C(v_{i-1}) - q_{\rho(v_i)}, & v_i \in \mathcal{V}^{del} \\ C(v_{i-1}), & \text{otherwise} \end{cases}$

When considering to insert a new service point u in S_w right after some existing point v_i, all the subsequent subpaths interconnecting consecutive nodes of S_w after v_i must be recomputed, to account for the updated departure and arrival times along Π_w. Due to the time-dependent nature of the travel-time metric, this is a non-trivial task to execute, prior to assessing the effectiveness of positioning the new event at a particular place within Π_w.

An instance of $\mathbb{VRP}_{\text{PDSTCtd}}$ is represented by a directed graph $G = (V, E)$, scalar arc-lengths $\lambda : E \mapsto \mathbb{R}_{>0}$, (periodic, continuous and piecewise-linear) arc-travel-time functions $(\tau_h[e] : [0, T) \mapsto \mathbb{R}_{>0})_{h\in H, e\in E}$, a sequence of requests $\mathcal{R}$, and a set of workers $\mathcal{W}$. As previously mentioned, we also construct the auxiliary graph G_{PD}. Because there may be nodes in $\mathcal{V}$ whose geo-locations do not coincide with vertices in V, some road-pedestrian connections are added between them by finding the nearest-neighbor pairs (x, v), for each $x \in V$ and $v \in \mathcal{V}$. The nearest-point search is done efficiently using an R-tree [6]. We also conduct sequentially shortest-path-tree computations for any involved vehicle type, to provide a set of (one-to-many) minimum-length paths and minimum-travel-time paths among geo-locations for elements of $V \cup \mathcal{V}$. For the (scalar) length metric we simply employ executions of Dijkstra's algorithm [4]. As for the time-dependent travel-time metric, the earliest-arrival-time computations are efficiently performed "on the fly", using the query algorithm `CFCA` of the CFLAT oracle [9] for time-dependent shortest paths, exactly when an arc in the PD graph G_{PD} is to be used by some worker. This is something that can be done efficiently by an online algorithm that only tries to fit into an existing solution a single new delivery request. On the contrary, the consideration of time-dependent travel-times renders impossible the construction of an exact MILP formulation; therefore, even the time-consuming construction of an optimal solution via MILP solvers becomes quite more challenging in this case. Section A in the appendix provides an approximate MILP formulation for $\mathbb{VRP}_{\text{PDSTCtd}}$ that considers some carefully selected scalar travel-time values for entire paths (rather than just arcs), exactly when they could be possibly used by some (any) feasible solution.

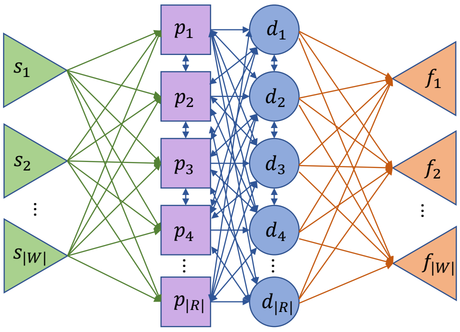


Figure 1 The pickup-and-delivery (PD) graph.

3 Insertion-based Schedulers for $\mathbb{VRP}_{\text{PDSTCtd}}$

The purpose of an insertion heuristic is to assign each request r to a worker w in a cost-optimal way, so that the new subtour S'_w (after adding the two service points of r) maintains the same relative order for the service points in S_w. The main reasons for such a requirement are simplicity and computational efficiency, since the consideration of all possible subtours for $R'_w = R_w \cup \{r\}$ would require the examination of an exponential number of subtours [14].

▶ **Definition 1** (Insertion-Based Heuristics). *Given a collection of subtours S_w for serving the subsets R_w of requests assigned to each operational worker $w \in \mathcal{W}$, and a new request r, an* Insertion-Based heuristic *determines for each $w \in \mathcal{W}$ a candidate subtour S'_w for $R'_w = R_w \cup \{r\}$, which achieves a minimum increase in w's contribution to some global-objective value, and leaves intact the relative order of the service points already in S_w. Eventually, r is assigned to the worker achieving the minimum increase, among all workers.*

The `Plain-Insertion` algorithm is well known in the literature of $\mathbb{VRP}$-related problems. A naïve implementation of such a heuristic would require quadratic, or even cubic computational time. A linear-time implementation of `Plain-Insertion` for $\mathbb{VRP}_{\text{PDSTC}}$ was recently proposed [16], which is based on a preprocessing step and on dynamic programming for computing the workers' scores (i.e., the marginal increases in cost if they were assigned the new request), for a scalar travel-time metric. We introduce in this section a variant of `Plain-Insertion`, called `TD-Insertion` for $\mathbb{VRP}_{\text{PDSTC}}$, which follows the main idea of the preprocessing of workers' paths in [16], so as to achieve early pruning of infeasible solutions, but with some major modifications so as to deal with the *time-dependent* travel-time metric and the consideration of earliest pickup-times for each request. The primary objective is to maximize the number of assigned requests. As a secondary objective, our algorithm considers two alternatives, as was previously explained: either the sum (i.e., ℓ_1-norm), or the sum-of-squares (i.e., ℓ_2-norm) of the workers' costs (distances, or travel-times). The ℓ_2-scoring criterion was inspired by [1], as an indirect means of *inducing* more balanced allocations of requests to the workers. We then incorporate in `TD-Insertion` the heuristic `Workload Balancer` (WB), which *enforces* some balance among the workers' assignments. We also consider a local-search improvement heuristic which post-processes the solutions provided by `TD-Insertion`, exploring among single-request relocation attempts for better solutions. Finally, we introduce `TD-Prophet`, a variant of `TD-Insertion` that, apart from actual requests, also includes in the produced subtours some short-term forecasts for future requests.

(a) Description of `TD-Insertion`

We denote as `TD-Insertion`$_{\kappa,\nu}$ the variant of `TD-Insertion` that assumes a cost metric $\kappa \in \{\tau, \lambda\}$ (τ for travel-times and λ for distances) and a norm $\ell_\nu \in \{\ell_1, \ell_2\}$ for assessing the scores of candidate insertion pairs of each new request. It is assumed inductively that: (i) each worker $w \in \mathcal{W}$ has already been assigned a subset of requests R_w, to be served according to the subtour (i.e., sequence) S_w of the corresponding service points; (ii) S_w has already been translated into some particular route Π^κ_w, depending on the particular cost metric $\kappa \in \{\tau, \lambda\}$ that we consider as primary. It should be noted at this point that Π^τ_w is indeed a route of minimum-travel-time interconnecting subtours. On the other hand, Π^λ_w is not necessarily a route of minimum-length interconnecting subtours. In particular, some minimum-travel-time interconnecting paths may also have been used in Π^λ_w for some pairs of consecutive points in S_w, only as contingency interconnecting routes for the case that the insertion of a service point with length-optimal interconnecting paths has lead to a violation of some temporal constraint of the subsequent service points. More about this issue is discussed in Subsection A.3.

Let $a(v_i)$ and $d(v_i)$ denote the arrival-time at v_i and the departure-time from v_i, respectively, as w moves along Π^κ_w. In a nutshell, the steps of `TD-Insertion`$_{\kappa,\ell_\nu}$ for $\kappa \in \{\tau, \lambda\}$ and $\nu \in \{1, 2\}$ are the following: For each new request r, at release-time t^{rel}_r, we test the insertion of r within the subtour $S_w = \langle v_0 = \chi^{start}_w, v_1, v_2, \ldots, v_{|S_w|-1} = \chi^{end}_w \rangle$ by iteratively placing the pickup node χ_r^{pic} right after position $i \in \{0 \ldots, |S_w| - 2\}$ and the delivery node χ_r^{del} right after position $j \in \{i, \ldots, |S_w| - 2\}$. We also require that $t_r^{rel} \leq d(v_{i+1})$, i.e., r cannot precede a service point whose departure time is already before r's release time. This insertion would result in the expanded subtour $S'_w = \langle v_0, v_1, \ldots, v_i, \chi_r^{pic}, v_{i+1}, \ldots, v_j, \chi_r^{del}, v_{j+1}, \ldots, v_{|S_w|-1} \rangle$ and the corresponding route Π'_w from Π_w^κ with the appropriate cost-optimizing interconnecting paths. For each pair (i, j) of candidate positions for the service points of r, a feasibility check of the spatial- and (time-dependent) temporal-constraints is performed along the suffix-subroute of Π'_w starting at node v_i. If all these service points are still feasible, then a marginal-increase value $Score_{\kappa,\lambda_\nu}(\Pi_w, i, j, r)$ is computed, to assess the impact on w's servicing cost of accepting the candidate positions (i, j) of S_w for serving r. In case of infeasibility, when $\kappa = \tau$ the candidate pair (i, j) is immediately rejected. When $\kappa = \lambda$, we alternatively construct the route Π''_w from Π_w with the appropriate travel-time-optimal interconnecting paths (only for the detours of r's service points). We provide now a detailed description of exactly how this is done.

(a.i) Preprocessing check-constraint indicators for candidate insertions

As in [16], given the (κ, ℓ_ν) pair of cost-metric and scoring-criterion that we consider, we use two check-constraint indicators for the service nodes of S_w, as w moves along the corresponding route Π_w (for simplicity, we slightly abuse notation by skipping the metric-dependent exponent, and the worker's shift-start-time):

- *slack*(v_i) is the *maximum tolerable time* for inserting a detour between v_i and v_{i+1}, without violating any of the (temporal) latest-delivery-times for nodes in $\Pi_w^{v_{i+1}:v_{|S_w|-1}}$.
- *ddl*(v_i) is an upper bound on the ultimate arrival-time at v_i so that neither the deadline for serving the request $\rho(v_i)$, nor the work-shift end of the carrying worker are violated.

For each arc $e = v_i v_{i+1} \in S_w \cap \mathcal{E}$, the travel-time $\tau[e](t)$, of the path $\pi_e = \langle v_i, v_{i+1} \rangle$ in G associated with e, can be either increased or decreased as a function of the departure-time t from v_i. Nevertheless, due to the FIFO property, the arrival-time function is non-decreasing: $\forall t < t',\ a[e](t) \leq a[e](t')$. Inserting χ_r^{pic} between v_i and v_{i+1} will give a new arrival-time $a'(v_{i+1}) \geq a(v_{i+1})$. Therefore, the current arrival-time value $a(v_{i+1})$ is a lower-bound whereas *slack*(v_i) is an upper bound on the arrival time at v_{i+1}, and their difference is an upper bound for the delay that may occur (due to some detours for adding new service points) between v_i and v_{i+1}. In order to incorporate earliest pickup-times in the *slack*(v_i) and *ddl*(v_i) indicators, the following dynamic-programming approach is adopted, as we move backwards along Π_w, from the end $v_{|S_w|-1} = \chi_w^{end}$ towards v_i:

1. *ddl*(v_i) values:
   - for the work-shift end node, $ddl(\chi_w^{end}) = t_w^{end}$;
   - for a commodity-delivery node $v_i \in \mathcal{V}^{del} \cap S_w$, $ddl(v_i) = t_{\rho(v_i)}^{ld}$;
   - for a commodity-pickup node $v_i \in \mathcal{V}^{pic} \cap S_w : v_j \in \mathcal{V}^{del} \wedge r = \rho(v_i) = \rho(v_j)$,

   $$ddl(v_i) = t_r^{ld} - \sum_{i \leq k < j} [(a(v_{k+1}) - d(v_k)) + t_k - b(v_k)]$$

   $$\text{where } t_k = \begin{cases} t_{\rho(v_k)}^{psrv}, & v_k \in \mathcal{V}^{pic} \\ t_{\rho(v_k)}^{dsrv}, & v_k \in \mathcal{V}^{del} \\ 0, & \text{otherwise} \end{cases}$$

2. *slack*(v_i) values: $slack(v_k) = \min\{\, ddl(v_{k+1}) - a(v_{k+1}), slack(v_{k+1}) + b(v_{k+1}) \,\}$ for k ranging from $|S_w| - 1$ down to i.

   Recall that $b(v)$ represents the required waiting-time at a service node v, or just the resulting idle-time (only at pickup nodes), when r imposes an earliest pickup-time t_r^{ep}.

It should be noted that, compared to a brute-force implementation of `Plain-Insertion`, the exploitation of these two auxiliary variables by our online algorithms allows the pruning without checking of many insertion-candidates, which has lead to a significant improvement of our implementations' execution times by at least 60%.

(a.ii) Efficient rejection of infeasible candidate insertions

The procedure is as follows: Assume for a new request r and a subtour S_w that we consider for insertion the candidate pair (i, j), for $0 \leq i \leq j \leq |S_w| - 2$. If $[a'(v_{i+1}) - a(v_{i+1}) > slack(v_i)] \vee [C(v_i) + q_r > Q_w]$ i.e., the resulting increase on the arrival-time at v_{i+1} exceeds the slack of v_i, or the resulting vehicle-load after picking up r at v_i causes a violation in the vehicle capacity, then the candidate pair (i, j) can be safely rejected. Moreover, if either of these two types of violation constraints occurs and $i < j$, i.e., only the pickup node of r is checked for insertion right after v_i, then all the candidate pairs $(i, m) : i \leq m \leq j$ can be safely rejected.

It should be noted at this point that, since the *slack times* are only upper-bounds, even if the above checks are passed, we still need to check for potential violations in latest-delivery-times of requests or in the work-shift end-time along the suffix of the new subtour S'_w that we create, under the time-dependent travel-time metric. Therefore, the time complexity to obtain all the feasible insertion-pairs for a new request along S_w is $O(|S_w|^2)$, due to the unavoidable time-dependent travel-time updates when checking for these potential violations.

(a.iii) Computation of scores for feasible candidate insertion-pairs per route Π_w

For a new request r, fix an arbitrary worker w whose vehicle-type is eligible for r: $H_w \in h_r$. Recall that we consider some arc-cost metric $\kappa \in \{\tau, \lambda\}$ (i.e., traversal-times, or distances) for all the arcs in the auxiliary PD graph. As for the assessment of the scores for candidate insertion pairs, as already mentioned, we consider that it is specified by the norm $\ell_\nu \in \{\ell_1, \ell_2\}$. For example, $\texttt{TD-Insertion}_{\tau,\ell_1}$ uses the travel-times cost metric for the routes and assesses the overhead of each candidate path according to the ℓ_1 norm, whereas $\texttt{TD-Insertion}_{\lambda,\ell_2}$ uses the arc-lengths metric for the routes and the overhead of each candidate path according to the ℓ_2 norm. For `TD-Insertion` we need to compute one of the following path-costs for worker w's subroute Π_w:

- For distance-metric: $Cost_\lambda(\Pi_w) = \sum_{e=uv\in\Pi_w}(\lambda_e)$.
- For travel-times metric: $Cost_\tau(\Pi_w) = \sum_{e=uv\in\Pi_w}[a(v) - d(u)]$.

Given Π_w and a particular candidate insertion pair (i, j) for a new request r, $\Pi_w(i, j, r)$ is the resultant candidate subroute from Π_w in which χ_r^{pic} is positioned right after v_i and χ_r^{del} is positioned right after v_j (and after χ_r^{pic}, in case that $i = j$). The *scores* (i.e., marginal costs) of this subroute are calculated as follows, for $\kappa \in \{\lambda, \tau\}$ and $\nu \in \{1, 2\}$:

$$Score_{\kappa,\ell_\nu}(\Pi_w, i, j, r) = \begin{cases} [Cost_\kappa(\Pi_w(i,j,r))]^\nu - [Cost_\kappa(\Pi_w)]^\nu, & \text{if } \Pi_w(i,j,r) \text{ is feasible} \\ \infty, & \text{if } \Pi_w(i,j,r) \text{ is infeasible} \end{cases}$$

The score of w for hosting r is then $Score_{\kappa,\ell_\nu}(w, r) = \min_{0\leq i\leq j\leq|S_w|-2} Score_{\kappa,\ell_\nu}(\Pi_w, i, j, r)$. Eventually, r is assigned to a worker $\hat{w}$ of minimum score: $\hat{w} \in \arg\min_{w\in\mathcal{W}} Score_{\kappa,\ell_\nu}(w, r)$.

Note that during step (a.ii), for each feasible pair (i, j), the score can be computed in parallel with the constraint-checking process. When a feasible pair (i, j) is verified, its (finite) score is compared to the minimum score discovered so far. Then, (i, j) is rejected immediately when the calculation of its score already gives a value larger than the current minimum score.

A particular attention should be given when working with distances: If some (i, j) leads to an eventually *infeasible route* $\Pi'_w := \Pi_w(w, i, j, r)$, because of violations of temporal constraints, then we repeat the process using travel-time-optimal (instead of distance-optimal) interconnecting paths for pairs of consecutive service points along Π_w (corresponding to arcs in the PD graph). This approach guarantees that, if there is any feasible solution at all for the new request, then the request will at least be allocated to some subroute Π''_w of finite score, even if having to use distance-suboptimal interconnecting paths. For `TD-Insertion` we considered two distinct contingency plans when facing such infeasibilities with the distance metric: (i) either construct Π''_w separately for each (i, j) whose Π'_w is time-infeasible, interconnecting with distance-suboptimal subtours r's service points within Π_w, or (ii) recompute from scratch an assignment for r, under the travel-time objective this time, but only when all the candidate pairs under the distance metric provided temporally infeasible routes. Eventually our decision for the experimental evaluation of `TD-Insertion` was to adopt the former contingency plan, as it adopts the travel-time metric not for each and every candidate pair but only for the problematic detours.

(b) Workload Balancer Heuristic (WB)

When running the experiments, it was observed that the optimal solutions provided by `TD-Insertion` involved only a few workers that shouldered the majority of the requests, while the rest of the workers did much less work, or were even not assigned any request at all. Towards providing more fair assignments for all the operational workers, we consider a threshold $\theta \geq 1$ and a penalty factor $\mu \geq 1$ and we introduce a bias for new requests in favor of workers with lighter (by means of traveled distance) workloads, even though some other workers might serve them with smaller marginal service costs. This bias is achieved by our `Workload Balancer` (WB) heuristic, which considers a slightly different scoring step for `TD-Insertion` for determining the winning worker per new request. In particular, upon the release of a new request r, let $\mathcal{W}_o$ be the set of the currently operational workers, and $Cost_{\kappa,\nu}(\Pi_w)$ be the cost of some worker $w \in \mathcal{W}_o$ for a given metric $\kappa \in \{\tau, \lambda\}$ and objective $\nu \in \{\ell_1, \ell_2\}$. The total cost of the current solution (before serving r) is $Cost_{\kappa,\nu}(\mathcal{W}_o) = \sum_{w \in \mathcal{W}_o} Cost_{\kappa,\nu}(\Pi_w)$. Then, each operational worker $w \in \mathcal{W}_o$ whose subroute-cost exceeds the average subroute-length in $\mathcal{W}_o$ by more than θ, gets a penalized score by a multiplicative factor $\mu > 0$: $\forall w \in \mathcal{W}_o,\ \forall \kappa \in \{\tau, \lambda\},\ \forall \nu \in \{\ell_1, \ell_2\}$,

$$Score^{wb}_{\kappa,\nu}(\Pi_w, i, j, r) := \left(1 + \mu \cdot \mathbb{I}_{\left\{Cost_{\lambda,\ell_1}(\Pi_w) > \theta \cdot \frac{Cost_{\lambda,\ell_1}(\mathcal{W}_o)}{|\mathcal{W}_o|}\right\}}\right) \cdot Score_{\kappa,\nu}(\Pi_w, i, j, r)$$

(c) Request Relocation Improvement Heuristic (RR)

A weakness of insertion-based heuristics is that they forbid changes in the assignment and the relative service order of the active requests, except for the new request. Towards amplifying this drawback, as in [7], we introduce the `Request-Relocation Improvement` (RR) heuristic, which conducts a sequence of local-search improvement attempts to the current solution as follows: For each $w \in \mathcal{W}_o$, and each $r \in \mathcal{R}_w$ that has *not* been picked up yet by w, $\chi^{pic}_r \in \mathcal{V}^{pic}$ and $\chi^{del}_r \in \mathcal{V}^{del}$ are removed from Π_w, making the appropriate shortcutting to Π_w so as to be a feasible subroute for $\mathcal{R}_w \setminus \{r\}$. By the FIFO property, this may cause no violation of a spatiotemporal constraint in Π_w and does not affect the routes of other workers. Consequently, r is relocated by `TD-Insertion`, either at a better position within Π_w or within the route of another operational worker.

(d) Digesting Demand Forcasts with TD-Prophet

In the typical online scenario, the request sequence $\mathcal{R}$ is initially unknown and is gradually revealed (per request) to the request scheduler. This knowledge gap is an important drawback for preparing a better and more organized scheduling plan. Apart from the higher risk of adopting suboptimal assignments, another significant burden is the necessity of the workers making large detours to serve newly revealed requests. Inspired by the Prophet-Insertion scheduler in [16], we introduce here a variant of TD-Insertion, called the TD-Prophet, which takes into account some sort of *short-term forecasts* for future requests and deals with them exactly as (virtual) requests with pickup/delivery points and spatiotemporal constraints. These virtual requests are a priori scheduled in the front of the request sequence to be handled by the scheduler, so as to be assigned to (initially idle) operational workers. This assignment is done using TD-Insertion. Consequently, their spatiotemporal constraints are deactivated (i.e., $q_r = 0, t_r^{ld} = \infty$), so as not to cause unnecessary infeasibilities for the workers' subroutes. The major difference of TD-Prophet from Prophet-Insertion in [16], apart of handling time-dependent travel-times, is that, after determining the assignment of the predictions to the workers, the delivery nodes of predictions of low appearance probability (below 80%) are removed (to better deal with any forecast inaccuracy). Also, when TD-Insertion decides to place the service nodes of a new request, say, at positions i and j respectively, any virtual pickup node (corresponding to a forecast) with low appearance probability (below 80%) between i and j is simply ignored. It should be noted that the demand-forecasting task is beyond the scope of this work and we consider this information to be provided as input to TD-Prophet. Nevertheless, Section B in the full version of the paper [10] describes exactly how this forecasting task is *simulated* for the real-world data set that we use for the needs of our experimentation.

4 Experimental Evaluation

We evaluated our algorithms using a real-world data set with records of pickup-and-delivery food and shopping orders during 3 consecutive working days, at the midium-sized city of Ptolemaida, Greece. In our experiments we assess the performance of our online schedulers for the actual request-sequence against two baseline solutions: (1) *human-curated solutions* provided by operators in the control room of a middleware platform mastering the service of food-order and shopping-delivery requests in Ptolemaida; and (2) optimal solutions to the *MILP formulation for a relaxation* (a carefully constructed instance of $\mathbb{VRP}_{\text{PDSTC}}$) of the actual instance of $\mathbb{VRP}_{\text{PDSTCtd}}$ (the detailed description of this relaxation, the proposed MILP formulation and the adopted solution method, are provided in Section A of the Appendix).

(a) Experimental Setup

The algorithms are in C++ (GNU GCC v.11.3.0). The experiments were conducted on an AMD EPYC 7552 48-Core 2.2GHz Processor with 256GB RAM and Ubuntu (22.04 LTS).

(b) Experimental Dataset

The dataset contains a pair $(\mathcal{W}, \mathcal{R})$ of a worker set and a request sequence, with actual pickup/delivery-times for the requests and work-shift intervals for the workers, within a period from Monday, July 3 2023 to Wednesday, July 5 2023, in the city of Ptolemaida in Northern Greece. All workers involved in this particular data set have used a single type of vehicle (scooters with a fixed-size storage). The human-curated service subtours were decided in real-time by well-experienced operators at the control center of a middleware platform providing couriers to food and shopping enterprises. The actual (GPS-recorded) service routes of the couriers were extracted from the pilot-phase event-logging database in the framework of a research project in which our group participated in the past [8]. These routes are already of high quality, since they were based on the long-term experience of the human operators, especially in the medium-size of the operational area (Ptolemaida). The construction of the road graph G was based on an OpenStreetMap dataset for Greece's road network [12]. The travel-time metric is provided by the OpenStreetMap service and the request-demand predictions were provided as input. G contains $|V| = 2547$ nodes and $|E| = 9514$ arcs. In the real data set some spatiotemporal restrictions were missing. In order to carry out a more realistic experimental evaluation, we adopted the following constraint scenario: Each request was assumed to have one-unit load and a duration of $40min$ between the latest-delivery-time and and the earliest-pickup-time, and service times of $1.5min$; and each worker uses a vehicle with a total capacity of 3 units. I.e., $\forall r \in \mathcal{R} \ \big(\ q_r = 1 \wedge t_r^{ld} = t_r^{ep} + 40min \wedge t_r^{psrv} = t_r^{dsrv} = 1.5min \ \big) \wedge \forall w \in \mathcal{W} \ \ (\ Q_w = 3 \ )$.

(c) Analysis of Experimental Results

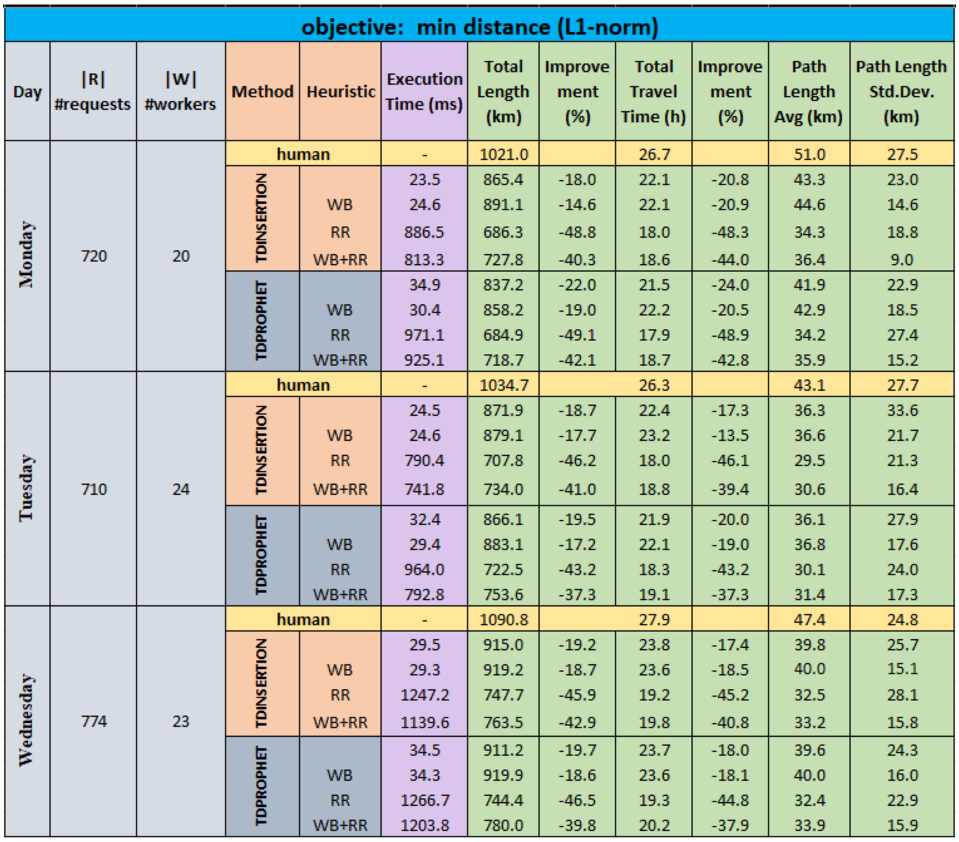
We executed three experiments, one per working day (Mon,Tue,Wed). Our online algorithms created the full sequences of worker subtours per day, starting from initially empty subtours. We experimented, exactly on the same instances, for an online algorithm $alg \in \{\texttt{TD-Insertion}_{\kappa,\nu}^{heur}, \texttt{TD-Prophet}_{\kappa,\nu}^{heur} \ : \ heur \in \{ \ \{ \ \}, \{wb\}, \{rr\}, \{wb, rr\} \ \}, \kappa \in \{\tau, \lambda\}, \nu \in \{\ell_1, \ell_2\}\}$ where $heur$ indicates whether specific heuristics are activated, κ determines the cost metric and ν specifies the type of the global objective.

Each variant of $\texttt{TD-Prophet}_{\kappa,\nu}$ works as follows: first we appended at the beginning of the request sequence a subset of predictions for virtual requests, which were then assigned to workers with $\texttt{TD-Insertion}_{\kappa,\nu}^{wb}$. The remaining sequence (of the real requests) were handled then sequentially, exactly as they appeared, by $\texttt{TD-Insertion}_{\kappa,\nu}$. Each real request was assumed to be visible to the scheduler only after its release time. Upon the release of a new (real) request $r \in \mathcal{R}$ at time t_r^{rel}, each variant of our algorithms executes the following substeps: $r \in \mathcal{R}$ is first assigned to a "moving" worker $w \in \mathcal{W}$ with the minimum score value, w.r.t. the objective function. Then, a detour event takes place, if the worker is instructed to change destination. Consequently, w's route is expanded by adding the service nodes of the new request. Finally, the new time-dependent interconnecting paths are computed, to (re)construct the route also covering the service points of the new request. As for preprocessing, in both the offline and the online scenarios, some common tasks are executed: (a) The preprocessing phase of CFLAT was executed by computing optimal trees, so that the interpolation of the travel-times at destinations constitutes an $(1 + \epsilon)$-approximation of the unknown time-dependent minimum-travel-time functions $\tau_h[o, d](t_o) : L \times V \times T \mapsto \mathbb{R}_{\geq 0}$, where $\epsilon = 0.1$, L is a subset of nodes (landmarks), $h = scooter$, and T is a one-week period. In principle we could use the query algorithm CFCA to approximately compute time-dependent distances "on the fly". Nevertheless, since the graph size is small, we set $L = V$ so as to avoid executing CFCA and to improve the approximation guarantees of the provided travel-time values. (b) Minimum distances $\lambda[o, d] : V \times V \mapsto \mathbb{R}_{\geq 0}$ are computed with Dijkstra calls from all the nodes in G_{PD}, under the distance metric λ.

The detailed presentation of the experimental results is deferred to the full version of the paper [10]. We demonstrate some indicative results in Table 1, which focuses on the distance metric and the ℓ_1 objective. The reported execution times of the online algorithms are average times per request. The table captures the resources spent: total travel-time (h) and total-length (km) traveled by the workers to serve requests, average (PathLen Avg) and variance (PathLen Var) of the workloads (measured in km) assigned to the workers. For the sake of a fair comparison, the human-curated subtours were translated into routes in such a way that all the interconnection paths are indeed distance-optimal paths, even if some of them are infeasible due to temporal constraint violations. This only works in favor of the baseline solutions. For the WB heuristic, we set $\theta = 1.5$ and $\mu = 2$. As shown in Table 1, against the quality of the human-curated assignments, there is a clear improvement of all variants of $\texttt{TD-Insertion}^{heur}_{\lambda,\ell_1}$, varying from 14.6% up to 49.1% decrease in total-length, and from 13.5% up to 48.9% decrease in total travel-time. The variants of $\texttt{TD-Prophet}^{heur}_{\lambda,\ell_1}$ provide an additional improvement over the corresponding variants $\texttt{TD-Insertion}^{heur}_{\lambda,\ell_1}$ by roughly 4%. The picture is similar also for the ℓ_2 objective, as shown in Table 2 in the full version of the paper [10]. Remarkably, ℓ_2 does not necessarily provide better solutions, but it guarantees much less variance in the workloads, without the need of the WB heuristic.

As for the solutions of the relaxed MILP formulation of $\mathbb{VRP}_{\text{PDSTCtd}}$, as shown in Table 5 in the full version of the paper, the involved solvers take hours to construct optimal solutions, even for small instances, whereas `TD-Insertion` finds very good solutions within amortized time per request that is smaller by several orders of magnitude. E.g., for 12 requests and 8 workers, the branch-and-cut method of SCIP spending up to 6 hours to find 31 feasible solutions and, via them, in the next phase, the best time-dependent metric converted solution which is only 0.64% better than the best of them in total-distance, within only 69ms per request.

Table 1 Experimentation of $\texttt{TD-Insertion}_{\lambda,\ell_1}$ and $\texttt{TD-Prophet}_{\lambda,\ell_1}$.



objective: min distance (L1-norm)

Day	|R| #requests	|W| #workers	Method	Heuristic	Execution Time (ms)	Total Length (km)	Improve ment (%)	Total Travel Time (h)	Improve ment (%)	Path Length Avg (km)	Path Length Std.Dev. (km)
Monday	720	20	human		-	1021.0		26.7		51.0	27.5
			TDINSERTION		23.5	865.4	-18.0	22.1	-20.8	43.3	23.0
				WB	24.6	891.1	-14.6	22.1	-20.9	44.6	14.6
				RR	886.5	686.3	-48.8	18.0	-48.3	34.3	18.8
				WB+RR	813.3	727.8	-40.3	18.6	-44.0	36.4	9.0
			TDPROPHET		34.9	837.2	-22.0	21.5	-24.0	41.9	22.9
				WB	30.4	858.2	-19.0	22.2	-20.5	42.9	18.5
				RR	971.1	684.9	-49.1	17.9	-48.9	34.2	27.4
				WB+RR	925.1	718.7	-42.1	18.7	-42.8	35.9	15.2
Tuesday	710	24	human		-	1034.7		26.3		43.1	27.7
			TDINSERTION		24.5	871.9	-18.7	22.4	-17.3	36.3	33.6
				WB	24.6	879.1	-17.7	23.2	-13.5	36.6	21.7
				RR	790.4	707.8	-46.2	18.0	-46.1	29.5	21.3
				WB+RR	741.8	734.0	-41.0	18.8	-39.4	30.6	16.4
			TDPROPHET		32.4	866.1	-19.5	21.9	-20.0	36.1	27.9
				WB	29.4	883.1	-17.2	22.1	-19.0	36.8	17.6
				RR	964.0	722.5	-43.2	18.3	-43.2	30.1	24.0
				WB+RR	792.8	753.6	-37.3	19.1	-37.3	31.4	17.3
Wednesday	774	23	human		-	1090.8		27.9		47.4	24.8
			TDINSERTION		29.5	915.0	-19.2	23.8	-17.4	39.8	25.7
				WB	29.3	919.2	-18.7	23.6	-18.5	40.0	15.1
				RR	1247.2	747.7	-45.9	19.2	-45.2	32.5	28.1
				WB+RR	1139.6	763.5	-42.9	19.8	-40.8	33.2	15.8
			TDPROPHET		34.5	911.2	-19.7	23.7	-18.0	39.6	24.3
				WB	34.3	919.9	-18.6	23.6	-18.1	40.0	16.0
				RR	1266.7	744.4	-46.5	19.3	-44.8	32.4	22.9
				WB+RR	1203.8	780.0	-39.8	20.2	-37.9	33.9	15.9

5 Concluding Remarks

In this paper we introduced, implemented and engineered two insertion-based online schedulers for the *time-dependent* variant $\mathbb{VRP}_{\text{PDSTCtd}}$ of $\mathbb{VRP}_{\text{PDSTC}}$, which were also experimentally evaluated on a real-world instance of food and shopping orders. In the future we plan to extend our online schedulers with more advanced local-search improvement heuristics, exploit them also by well known metaheuristics that are efficient for $\mathbb{VRP}$, such as the `ALNS` metaheuristic, and also to explore in more depth offline solvers which are custom-tailored to this particular problem.

References

1 Tolga Bektas and Adam Letchford. Using ℓ^p-norms for fairness in combinatorial optimisation. *Computers & Operations Research*, 120(104975), 2020. `doi:10.1016/j.cor.2020.104975`.

2 K. Bestuzheva, M. Besançon, W. K. Chen, A. Chmiela, T. Donkiewicz, J. van Doornmalen, L. Eifler, O. Gaul, G. Gamrath, A. Gleixner, L. Gottwald, C. Graczyk, K. Halbig, A. Hoen, C. Hojny, R. van der Hulst, T. Koch, M. Lübbecke, S. J. Maher, F. Matter, E. Mühmer, B. Müller, M. E. Pfetsch, D. Rehfeldt, S. Schlein, F. Schlösser, F. Serrano, Y. Shinano, B. Sofranac, M. Turner, S. Vigerske, F. Wegscheider, P. Wellner, D. Weninger, and J. Witzig. The SCIP Optimization Suite 8.0. Technical report, Optimization Online, 2021.

3 Peng Cheng, Hao Xin, and Lei Chen. Utility-aware ridesharing on road networks. *SIGMOD*, pages 1197–1210, 2017. `doi:10.1145/3035918.3064008`.

4 Edsger Wybe Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959. `doi:10.1007/BF01386390`.

5 Gurobi Optimization, LLC. Gurobi Optimizer v.10.0.2 , 2023. URL: `https://www.gurobi.com`.

6 Antonin Guttman. R-trees: A dynamic index structure for spatial searching. *SIGMOD*, 14(2):47–57, 1984. `doi:10.1145/971697.602266`.

7 Penny Holborn, Jonathan Thompson, and Rhyd Lewis. Combining heuristic and exact methods to solve the vehicle routing problem with pickups, deliveries and time windows. In *Evolutionary Computation in Combinatorial Optimization: 12th European Conference (EvoCOP 2012)*, pages 63–74. Springer, 2012. `doi:10.1007/978-3-642-29124-1_6`.

8 iDeliver - delivering platform for delivering goods as a service. `https://i-deliver.gr`, 2023.

9 Spyros Kontogiannis, Georgia Papastavrou, Andreas Paraskevopoulos, Dorothea Wagner, and Christos Zaroliagis. Improved Oracles for Time-Dependent Road Networks. In *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017)*, volume 59 of *OpenAccess Series in Informatics (OASIcs)*, pages 4:1–4:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. `doi:10.4230/OASIcs.ATMOS.2017.4`.

10 Spyros Kontogiannis, Andreas Paraskevopoulos, and Christos Zaroliagis. Online Vehicle Routing with Pickups and Deliveries under Time-Dependent Travel-Time Constraints. Arxiv technical report, ArXiv, 2024. `doi:10.48550/arXiv.2408.06324`.

11 Spyros Kontogiannis, Dorothea Wagner, and Christos Zaroliagis. An axiomatic approach to time-dependent shortest path oracles. *Algorithmica*, 84:815–870, 2022. `doi:10.1007/s00453-021-00922-8`.

12 OpenStreetMap datasets. `https://download.geofabrik.de/europe.html`, 2023.

13 Manfred Padberg and Giovanni Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, 33(1):60–100, 1991. `doi:10.1137/1033004`.

14 J. Pan, G. Li, and J. Hu. Ridesharing: Simulator, benchmark, and evaluation. *PVLDB*, 12:1085–1098, 2019. `doi:10.14778/3339490.3339493`.

15 Solving Constraint Integer Programs. Scip optimization suite v.8.0.3, 2023. URL: `https://scipopt.org`.

16 Yongxin Tong, Yuxiang Zeng, Zimu Zhou, Lei Chen, and Ke Xu. Unified route planning for shared mobility: An insertion-based framework. *ACM Transactions on Database Systems*, 47:2:1–2:48, 2022. `doi:10.1145/3488723`.

A Construction of a Relaxed MILP Formulation for $\mathbb{VRP}_{\text{PDSTCtd}}$

Recall that each assignment of requests to workers for an instance of $\mathbb{VRP}_{\text{PDSTCtd}}$ is represented as a collection $\{S_w : w \in \mathcal{W}\}$ of vertex-disjoint (simple) paths in $G_{PD} = (\mathcal{V}, \mathcal{E})$. Of course, in order to actually have a feasible solution in the end, we should translate each assignment S_w into a (not necessarily simple) walk Π_w to be followed by w in the underlying road network $G = (V, E)$, by substituting each pair of consecutive points in S_w (i.e., an arc in G_{PD}) with some cost-minimal interconnecting path of G. At this point, there are two options for the interconnection of the endpoints of each arc $e = uv \in \mathcal{E}$: Use in the road network G either a travel-time-optimal (and distance-suboptimal) (u, v)-path, or a distance-optimal (and travel-time-suboptimal) (u, v)-path. When the cost-objective is based on the travel-time metric, all interconnecting paths for arcs of $\mathcal{E}$ are naturally minimum-travel-time paths in G, mainly due to the FIFO property of the metric. On the other hand, when the cost-objective is based on the distance metric, although the interconnecting paths for arcs of $\mathcal{E}$ should ideally be minimum-distance paths in G, such a choice might lead to infeasible solutions. We explain in subsection A.3 how we resolve this issue in such a way that, to the least, whenever there is a feasible solution of a given maximum number of serviced requests, one such solution should be found (even if it is suboptimal in the cost-objective).

Before that, we first consider a simplified situation (cf. A.1) where the travel-time metric consists of scalar values for the arcs in G (i.e., it is time-independent). We then perceive all the temporal parameters (e.g., travel-times, arrival-times) as scalars, which can be precomputed. Abusing slightly the notation for the sake of simplicity, we use only the names of the temporal functions, without their explicit dependence on departure-time values from the tail of an arc or from the origin of a path, as the corresponding constants. For example, we write $\tau[\Pi_w]$ for the scalar approximation of the time-dependent path-travel-time function $\tau[\Pi_w](t_w^{start})$. Given those (constant) travel-time values, in subsection A.2) we provide a MILP formulation for the (time-independent) relaxation of the actual instance of $\mathbb{VRP}_{\text{PDSTCtd}}$ that we wish to solve. This MILP is then fed to several MILP solvers for providing an offline solution (cf. A.4), to act as alternative baseline solutions for quantifying the quality of the provided heuristic solutions by our online solvers. Of course, even the solutions provided by the offline solvers for the MILP relaxation are suboptimal solutions to the $\mathbb{VRP}_{\text{PDSTCtd}}$ instance at hand. The challenge is exactly to adopt a time-independent travel-time metric which is somehow more informative than just considering the average, or the freef-flow arc-traversal times of the road segments and thus renders offline solutions closer to optimality.

A.1 Approximating Time-Dependent Travel-Times

This subsection concerns the determination of scalar travel-time values to all the arcs in the PD-graph G_{PD}, when the global objective to consider is the minimization of (sums, or sums-of-squares of) travel-time along the actual servicing paths of the workers. In this case, each arc of G_{PD} actually represents a minimum-travel-time interconnecting path for its endpoints, in the underlying road network G.

Rather than simply considering only average (or, free-flow) travel-time values per arc in G and then conducting shortest-path computations between the endpoints of arcs in G_{PD}, as is typically the case, we construct a more meaningful static travel-time metric for the arcs of G_{PD}, which tries to be as close as possible to the actual time-dependent travel-time metric in G, taking into account that specific connections may only appear at specific parts of the workers' subtours. In particular, we construct relaxated time-indepdenent travel-times for all the arcs in G_{PD} in three consecutive phases,

Phase 1: We compute the actual earliest arrival-times (and thus, also the minimum travel-times) from the starting location of each worker to every of the pickup point, under the time-dependent metric. In particular, fix an arbitrary arc $e = (u = \chi_w^{start}, v = \chi_r^{pic}) \in \mathcal{V}^{start} \times \mathcal{V}^{pic}$ from some worker's shift-start point to some request's pickup point. The departure-time is definitely t_w^{start}. Therefore, using the query algorithm `CFCA` of the `CFLAT` oracle, we can determine a $(1+\epsilon)$-approximation of the minimum travel-time value $\tau[\Pi_w^{u:v}](t_w^{start})$. The eventual scalar travel-time approximation for e (when considered as candidate for first arc in some subtour) is defined as $\tau_{e,h_w} = \tau_{h_w}[\Pi_w^{u:v}](t_w^{start}) + t_r^{psrv}$, i.e., we add to the actual travel-time value the service-time at the pickup point χ_r^{pic}.

Phase 2: We consider all the arcs of G_{PD} emanating from pickup points, towards other pickup points or delivery points (acting as candidates for second, or even later arcs within subtours). To compute scalar approximations of their travel-times, we make calls of `CFCA` from each pickup point χ_r^{pic} and each vehicle-type $h \in H_r$, towards all destinations in $\mathcal{V}^{pic} \cup \mathcal{V}^{del}$. As departure-times from χ_r^{pic} we consider the maximum of its earliest pickup-time t_r^{ep} and its earliest arrival-time from any worker with the specific vehicle type. The resulting (time-dependent) minimum travel-time values at the destinations, plus the service times at the destinations, determine the scalar approximations $\tau_{e,h}$, for all the arcs $(\chi_r^{pic}, v) \in \mathcal{E} \cap \mathcal{V}^{pic} \times (\mathcal{V}^{pic} \cup \mathcal{V}^{del})$ and vehicle types $h \in H_r$.

Phase 3: We consider all the arcs of G_{PD} emanating from delivery points, towards other pickup points, delivery points, or work-shift ending points (as candidates for third, or even later arcs within subtours). To compute scalar approximations of their travel-times, we make again calls of `CFCA` from any delivery point $\chi_r^{del} \in \mathcal{V}^{del}$ towards all destinations in $v \in \mathcal{V}^{pic} \cup \mathcal{V}^{del} \cup \mathcal{V}^{end}$. As departure-time from χ_r^{del} we consider the earliest arrival-time, among all eligible vehicle types $h \in H_r$, from the corresponding pickup point χ_r^{pic}, as it was computed in the second phase, since it definitely has to precede that delivery point. The resulting minimum arrival-times computed by these calls plus the service times (if any) at the destinations, determine the scalar approximations $\tau_{e,h}$ of the travel-times that we consider, for all the arcs $(\chi_r^{del}, v) \in \mathcal{E} \cap \mathcal{V}^{del} \times (\mathcal{V}^{pic} \cup \mathcal{V}^{del} \cup \mathcal{V}^{end})$ and eligible vehicle-type $h \in H_r$.

A.2 MILP Formulation for Relaxation of $\mathbb{VRP}_{\text{PDSTCtd}}$

With the above mentioned static travel-time metric at hand, we may proceed with the construction of the relaxed MILP formulation of $\mathbb{VRP}_{\text{PDSTCtd}}$. Recall that each arc $(\chi_{w_1}^{start}, \chi_{r_1}^{pic}) \in \mathcal{V}^{start} \times \mathcal{V}^{pic}$ and each arc $(\chi_{r_2}^{del}, \chi_{w_2}^{end}) \in \mathcal{V}^{del} \times \mathcal{V}^{end}$ may be "traversed" by workers w_1 and w_2 if and only if r_1 and r_2 were assigned to them, respectively. The rest of the arcs in $\mathcal{E}$ may be "traversed" by any worker whose vehicle is eligible for the serviced requests at its endpoint(s). Therefore, some binary decision variables are employed to indicate the traversal of arcs by workers and the assignment of requests to workers: For each arc $e \in \mathcal{E} \cap ((\mathcal{V}^{pic} \cup \mathcal{V}^{del}) \times (\mathcal{V}^{pic} \cup \mathcal{V}^{del}))$, each request $r \in \mathcal{R}$, and each worker $w \in \mathcal{W}$, $x_{e,w}$ indicates whether w traverses e, and $x_{r,w}$ indicates whether r is assigned to w.

We proceed with the definition of some constants for earliest arrival-times at nodes in $\mathcal{V}$ and modifications in vehicle-loads, as some worker w traverses an arc $e = uv \in S_w$ towards a service node $v \in \mathcal{V}^{pic} \cup \mathcal{V}^{del}$. Recall that any feasible solution is a collection of subtours $\{S_w : w \in \mathcal{W}\}$ which correspond to vertex-disjoint paths in G_{PD}. In particular, for each edge $e = uv \in \mathcal{E}$, the constant q_e represents the change in a vehicle's load, when traversing e: if $v = \chi_r^{pic} \in \mathcal{V}^{pic}$ then $q_e = q_r$; if $v = \chi_r^{del} \in \mathcal{V}^{del}$, then $q_e = -q_r$; otherwise, $q_e = 0$. Moreover, each request $r \in \mathcal{R}$ comes with a (large positive) profit σ_r, to be considered only when r is assigned to some worker for its service. Finally, for each node $v \in \mathcal{V}$, the continuous variable a_v captures either the arrival-time of the unique worker (if any) who serves the corresponding request r along its assigned route, when $v \in \{\chi_r^{pic}, \chi_r^{del} \subseteq \mathcal{V}^{pic} \cup \mathcal{V}^{del}$; or a worker's shift starting time t_w^{start}, when $v = \chi_w^{start} \in \mathcal{V}^{start}$ is its shift-starting node; or, the eventual arrival-time at the end of the entire route Π_w of a worker, when $v = \chi^{end} \in \mathcal{V}^{end}$ is its shift-ending node. Along each arc $e = uv \in \mathcal{E}$, the values of the variables a_u and a_v should be compliant with the required time for the moving worker (with a particular vehicle type) to traverse e.

For the sake of simplicity, we make here the assumption that all workers possess the same vehicle type, as is the case in our real-world data set. The proposed relaxed mixed integer linear program (MILP) for $\mathbb{VRP}_{\textsc{PDSTCtd}}$ is shown in Figure 2. It can be easily extended to also cover the case of more vehicle types for the workers.

$$
\begin{array}{ll}
\min \displaystyle\sum_{e\in\mathcal{E}, w\in\mathcal{W}} c_{e,w} x_{e,w} - \sum_{r\in\mathcal{R}} \sigma_r \sum_{w\in\mathcal{W}} x_{r,w} & \\
\text{s.t.} & \\
0 \le \displaystyle\sum_{v\in\mathcal{V}^{pic}} x_{uv,\gamma(u)} \le 1, \forall u \in \mathcal{V}^{start} & (1), \\
\displaystyle\sum_{uv\in\mathcal{E}: u=\chi_r^{pic}} x_{uv,w} = x_{r,w}, \quad \forall r \in \mathcal{R}, w \in \mathcal{W} & (2), \\
\displaystyle\sum_{uv\in\mathcal{E}: v=\chi_r^{pic}} x_{uv,w} = x_{r,w}, \quad \forall r \in \mathcal{R}, w \in \mathcal{W} & (3), \\
\displaystyle\sum_{uv\in\mathcal{E}: u=\chi_r^{del}} x_{uv,w} = x_{r,w}, \quad \forall r \in \mathcal{R}, w \in \mathcal{W} & (4), \\
\displaystyle\sum_{uv\in\mathcal{E}: v=\chi_r^{del}} x_{uv,w} = x_{r,w}, \quad \forall r \in \mathcal{R}, w \in \mathcal{W} & (5), \\
0 \le \displaystyle\sum_{w\in\mathcal{W}} x_{r,w} \le 1, \quad \forall r \in \mathcal{R} & (6), \\
t_v^{start} \le a_v \le t_v^{end}, \quad \forall v \in \mathcal{V} & (7), \\
a_{\chi_r^{pic}} \le a_{\chi_r^{del}}, \quad \forall r \in \mathcal{R} & (8), \\
|a_v - a_u - \displaystyle\sum_{w\in\mathcal{W}} (T_{max} + \tau_{uv,w}) x_{uv,w}| \le T_{max}, \forall uv \in \mathcal{E} & (9), \\
|q_v - q_u - \displaystyle\sum_{w\in\mathcal{W}} (Q_{max} + q_{uv}) x_{uv,w}| \le Q_{max}, \forall uv \in \mathcal{E} & (10), \\
0 \le q_v \le \displaystyle\sum_{r\in\mathcal{R}, w\in\mathcal{W}} x_{r,w} Q_w, \quad \forall v \in \mathcal{V}_{\mathcal{P}} & (11), \\
x_{e,w}, x_{r,w} \in \{0,1\}, \quad \forall e : uv \in \mathcal{E}, w \in \mathcal{W}, r \in \mathcal{R} & (12), \\
q_v = 0, v \in \mathcal{V}^{start} \cup \mathcal{V}^{end}; \quad 0 \le q_v \le Q_{max}, v \in \mathcal{V}^{pic} \cup \mathcal{V}^{del} & (13)
\end{array}
$$

Figure 2 The relaxed MILP formulation of $\mathbb{VRP}_{\textsc{PDSTCtd}}$.

The objective function seeks as a primary goal to maximize the number of served requests (recall the large positive values for the σ_r parameters) and, as a secondary goal, to minimize the aggregate travel cost for having the selected requests served by the workers. Towards this direction, we construct the objective as the sum of two terms. The first term accounts for the aggregate cost to serve all the accepted requests, as determined by the sum of costs $c_{e,w} \cdot x_{e,w}$ for those arcs of G_{PD} which are used in the solution. The value of the coefficient $c_{e,w}$ depends on the metric that is considered for the objective, i.e., it is associated with either the arc-length λ_e, or with the (approximate) arc-travel-time $\tau_{e,w}$ (but excluding the embedded service times). The second term determines the negative of the aggregate profit for serving requests: For each request $r \in \mathcal{R}$, the coefficient σ_r denotes the "profit" for having r served by some worker. These coefficients are set to a sufficiently large value (based on an upper bound to the worst possible path-travel-time or to the maximum length from any origin towards any destination), so as to enforce the service of as many requests as possible. The sum of profits for all the served requests is then subtracted from the overall service cost.

As for the constraints of the MILP: (1) ensures that any worker's subtour may start with a move towards at most one pickup point. (2-5) enforce that each worker may depart from / enter the pickup / delivery) point of some request towards / from any other node, only if the corresponding request is assigned to her. (6) ensures that any request r is served by at most one worker. (7) enforces the arrival time a_v at each node v is in the allowable time window, where: (a) for $v \in \{\chi_w^{start}, \chi_w^{end}\}$, $t_v^{start} = t_w^{start}$ and $t_v^{end} = t_w^{end}$; and (b) for $v \in \{\chi_r^{pic}, \chi_r^{del}\}$, $t_v^{start} = t_r^{ep}$ and $t_v^{end} = t_r^{ld}$. (8) ensures that the pickup-time point of r precedes the delivery-time point of r. (9) stipulates that if an arc $e = uv\mathcal{E}$ is traversed by some worker w, then the arrival time a_v at v must be the result of the arrival-time a_u at u plus w's travel-time $\tau_{e,w}$ along e. (10) stipulates that if an arc $e = uv \in \mathcal{E}$ is traversed by some worker w, then the change of w's vehicle-load q_v at v results from the vehicle-load q_u at u plus the pickup-load / minus the delivery-load $q_{\rho(v)} = |q_e|$ that corresponds to the request $\rho(v)$. (11) ensures that the vehicle's load at any pickup-node v assigned to w, never exceeds the vehicle's maximum capacity Q_w. The constants T_{max} and Q_{max} are the result of applying the *big-M linearization* method, and their values, in direct dependence on the problem instance, are selected as the maximum distance $|a_v - a_u|$, $|q_v - q_u|$, $\forall uv \in \mathcal{E}$, e.g. a loose bound could be $T_{\max} = \max_{w \in \mathcal{W}}(t_w^{end} - t_w^{start})$ and $Q_{\max} = \max_{w \in \mathcal{W}} Q_w$. Note that in (9) and (10) only the lower bounds are tight; the upper bounds are relaxed, T_{max} and Q_{max} theoretically can also be perceived as ∞. Their actual values are determined in relation to the rest of the constraints. This is done on purpose, especially for the arrival decision variables, because those variables in pickup-nodes have to be increased to reach the earliest pickup-times in the case of a non-zero buffer time.

A.3 Translating Assignments to Routes under the Distance Metric

It is important to note at this point that for the MILP formulation provided in Figure 2, which was constructed on top of the PD graph with arc-costs equal to the minimum travel-times of their endpoints in the underlying road graph G, the optimal solutions indeed maximize the aggregate profit for serving the accepted requests and, at the same time, minimize the aggregate service cost for having the workers on the move, when the cost for each arc $uv \in \mathcal{E}$ is indeed measured by the total travel-time of the workers from u to v in the underlying road graph G.

Unfortunately, when the secondary objective (the aggregate service cost) is measured by the total distance traveled by the workers, it is no longer true that the consideration of a route of consecutive distance-optimal paths for implementing a given subtour is the right choice for implementing a worker's subtour. This then causes a crucial dilemma: *which weights should be considered for the arcs of $\mathcal{E}$?* For example, in Figure 3 the worker can move from u to v along one of two uv-routes in the underlying road network G, but the upper route is distance-optimal but time-infeasible and the lower route is travel-time optimal but distance-suboptimal. Then, for the unique arc $uv \in \mathcal{E}$, using the distance-optimal (but travel-time-suboptimal) weights would lead to a MILP formulation where the single request is impossible to serve, whereas using the travel-time-optimal (but distance-suboptimal) weights for the arcs in the PD graph would certainly provide a feasible solution, whose distance-related service cost may be far from being optimal. Recall that our primary objective is to have a maximum profit by the accepted requests for service (e.g., to have as many requests served, as possible). Nevertheless, we wish (given that) to move towards optimizing also the distance-related service cost, even though we cannot possibly reach it. Towards this direction, we change the PD graph as follows: For each $e = uv \in \mathcal{E}$, we attach two Pareto-optimal routes in the underlying road graph G, a distance-optimal (but travel-time-suboptimal) uv-route $\pi^{l}_{u,v}$, and a travel-time-optimal (but distance-suboptimal) uv-route $\pi^{\tau}_{u,v}$, per type of vehicle. This way, our (updated for the distance-optimal service cost objective) MILP formulation tries to find either a distance-optimal and spatiotemporally feasible solution, or at least a distance-suboptimal feasible solution that employs the cheapest (w.r.t. extra distance to be traveled) subset of travel-time optimal connections so as to guarantee feasibility. Of course, this is still not the required distance-optimal solution for the maximum profit for the accepted requests, because it might be the case that some connecting paths in the road graph which are suboptimal for both distance and travel-time criteria might be preferable. Nevertheless, the primary goal of maximing the profit of accepted requests is now achieved.

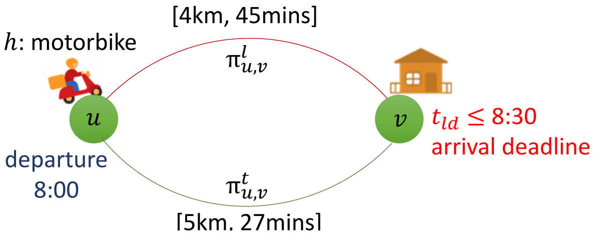


Figure 3 Finding a feasible solution to minimize distance while respecting the time constraints. The worker arrives at u at 8:00. The arrival-time at node v via the distance-optimal path $\pi^{l}_{u,v}$ is 8:45, i.e., too late w.r.t. the latest-delivery-time deadline (8:30). On the other hand, the arrival-time at v via the travel-time optimal path $\pi^{t}_{u,v}$ is 8:27, i.e., catching up the delivery deadline, at the cost of a slightly longer distance to travel.

A.4 Offline Solvers for Relaxed MILP Formulation of $\mathbb{VRP}_{\mathrm{PDSTCtd}}$

The core method used for solving the MILP formulation of $\mathbb{VRP}_{\mathrm{PDSTC}}$ is based on built-in implementations in SCIP [2, 15] and Gurobi [5] of the *branch-and-cut* method [13]. Since the produced solution has taken into account, not the actual time-dependent travel-time metric, but a time-independent approximation metric for it, as was already explained in Section A.1, we have to cross-check that the produced solution indeed respects all the spatiotemporal constraints of the instance. This is done as follows: We examine up to 10 of the best feasible solutions found from the MILP solver. Each solution is a set of subtours $\{S_w : w \in \mathcal{W}\}$. For each worker $w \in \mathcal{W}$ and the corresponding subtour S_w which dictates the visiting order of the service points for all the requests assigned to her, we recompute the (now time-dependent) optimal interconnecting paths for consecutive points of S_w. The spatial constraints (relating to vehicle capacities) are certainly preserved. As for the temporal constraints, we recheck along the subtour if any arrival-time at a delivery service or shift-ending node now violates a temporal constraint, thus rendering the particular subtour infeasible. In such a case, we try to "repair" the route of S_w in the following way: at each $v_x \in \Pi_w \cap \{\mathcal{V}^{del} \cup \mathcal{V}^{end}\}$, where $\alpha(v_x) > t^{ld}_{\rho(v_x)}$ or $\alpha(v_x) > t^{end}_w$ (i.e. a latest delivery or shift-ending deadline is violated), we traverse the route $\Pi_w^{v_1:v_x}$ backwards, i.e. from v_x up to v_1, and successively any contained intermediate subpath in Π_w that was computed for length-minimization is replaced by a corresponding optimal subpath that minimizes the travel-time. If there is no length-optimal path or the applied replacements eventually are not enough to deal with the deadline violations, then the whole solution is rejected, and an additional constraint to block the selection of the infeasible subtours in G_{PD} is added to the MILP formulation (Figure 2). This process is applied to any MILP solver's examined feasible solution. At the end, if there is no time-dependent-metric converted feasible solution, then there is the possibility to use the new MILP formulation (with the added constraints) to solve the problem again for computing new solutions that will hopefully overcome the deadline violations.

Periodic Timetabling: Travel Time vs. Regenerative Energy

Sven Jäger
RPTU Kaiserslautern-Landau, Germany

Sarah Roth
RPTU Kaiserslautern-Landau, Germany

Anita Schöbel
RPTU Kaiserslautern-Landau, Germany
Fraunhofer Institute for Industrial Mathematics, Kaiserslautern, Germany

Abstract

While it is important to provide attractive public transportation to the passengers allowing short travel times, it should also be a major concern to reduce the amount of energy used by the public transport system. Electrical trains can regenerate energy when braking, which can be used by a nearby accelerating train. Therefore, apart from the minimization of travel times, the maximization of brake-traction overlaps of nearby trains is an important objective in periodic timetabling. Recently, this has been studied in a model allowing small modifications of a nominal timetable. We investigate the problem of finding periodic timetables that are globally good in both objective functions. We show that the general problem is NP-hard, even restricted to a single transfer station and if only travel time is to be minimized, and give an algorithm with an additive error bound for maximizing the brake-traction overlap on this small network. Moreover, we identify special cases in which the problem is solvable in polynomial time. Finally, we demonstrate the trade-off between the two objective functions in an experimental study.

2012 ACM Subject Classification Applied computing → Transportation

Keywords and phrases periodic timetabling, regenerative braking

Digital Object Identifier 10.4230/OASIcs.ATMOS.2024.10

Funding The first two authors were funded by the German Federal Minisry of Education and Research (BMBF) (Project number 05M22UKB).

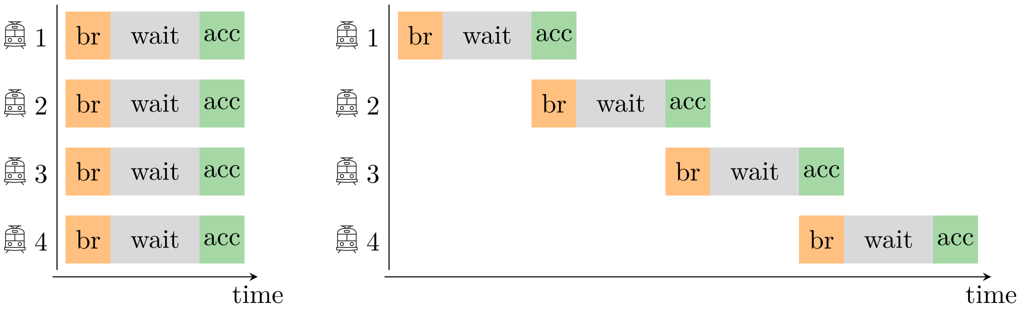
Acknowledgements We want to thank Niels Lindner for fruitful discussions about the topic.

1 Introduction

In order to reach the climate goals, it is necessary to strengthen the role of public transportation in passenger transport. However, also the public transport system itself consumes a large amount of energy. Modern electric motors are able to regenerate energy while braking. In the context of rail traffic, the most efficient way to use the regained energy is to transfer it via the catenary to an accelerating train close by. Therefore, it is sensible to schedule train timetables in a way that synchronizes braking and acceleration processes of nearby trains. Such a schedule has two advantages concerning the energy usage. First, it enables a maximum usage of the regenerated energy and, hence, reduces the total amount of energy that needs to be bought by the public transport company. Second, it prevents power peaks that might surcharge the transportation system's power supply.

However, from a passenger perspective, this synchronization of braking and acceleration processes of two trains is the worst possible case as it prevents a passenger transfer from the braking to the accelerating train. Narrowly missing a train leads to frustration of the passengers and long waiting times might cause them to choose the car over public transport.

24th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2024).
Editors: Paul C. Bouman and Spyros C. Kontogiannis; Article No. 10; pp. 10:1–10:20
OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



(a) arr/dep around fixed time. **(b)** Simultaneous braking/accelerating of pairs of trains.

Figure 1 Timetable patterns of braking, waiting and acceleration phases of four trains.

As an illustration of this trade-off, we consider an example motivated by Swiss railways. Here, the operated timetable prevents such situations of narrowly missing a train. The trains are scheduled in a regular interval timetable. At each station there is a fixed time. Shortly before this time all trains stopping at the station arrive, and the trains depart shortly after that time, see Figure 1a. This enables short transfer times to all directions. On the other hand, a transfer of regenerative braking energy from one train to another is impossible. For this objective, an efficient timetable would schedule the trains one after another such that the braking and acceleration phases overlap pairwise, see Figure 1b.

While it is beneficial to the environment to use as much of the regenerative energy as possible, it is also of utmost importance to provide attractive public transportation to the passengers. In this paper, we investigate a bicriteria problem with the aims to maximize the brake-traction overlap enabling the usage of regenerative energy and to minimize the passengers' travel times. We study this problem in the periodic version, where all train lines are operated repeatedly with a fixed period time.

Related Work

The task of designing efficient railway timetables has been subject to study at least since 1989 [12]. Traditionally, the literature on timetabling focuses on minimizing the passengers' travel time. An overview can be found in [7]. As mentioned above, the increasing importance of saving energy has sparked significant research efforts towards this goal in the engineering sciences. A complete review of all these works goes far beyond the scope of this paper. Instead, we only mention some particularly important papers and refer to the survey by Scheepmaker, Goverde, and Kroon [11] and the exemplary recent papers [8, 6, 13], which contain more extensive literature reviews.

There are two ways in which the timetable can affect the trains' energy consumption. On the one hand, there is the idea of saving energy by the implementation of energy-efficient driving strategies [5]. These depend on the time scheduled for each driving section; typically longer travel times require less energy. Ghoseiri, Szidarovszky, and Asgharpour [3] considered a multi-objective train scheduling model, combining the objectives of minimizing energy and minimizing travel time, and approximate the Pareto frontier using the ε-constraint method.

On the other hand, the timetable can influence the usage of regenerative energy in train systems. This was first researched by Ramos Pena, Fernández, and Cucala [10], who allow a modification of the dwell times to increase the brake-traction overlap. A more detailed modelling of the energy consumption that combines the driving strategies and the brake-traction overlaps has been studied by Yin, Yang, Tang, Gao, and Ran [17], who devised a

Lagrangian relaxation-based heuristic for this problem. In the other direction, Gupta, Tobin, and Pavel [2] considered a very simplified linear programming model to synchronize the start times of braking phases and the end times of acceleration times.

The bicriteria problem of minimizing the passenger travel times and maximizing the brake-traction overlap has been investigated by Yang, Ning, Li, and Tang [16], who developed a genetic algorithm for it. Moreover, Yang, Liao, Wu, Timmermans, Sun, and Gao [15] apply the NSGA-II algorithm for approximating the Pareto frontier.

All these works considered given aperiodic timetables that can be modified. Only recently, the study of the periodic version of this problem was initiated by Wang, Zhu, and Corman [14]. They assume a given nominal periodic timetable and develop a first model that can be used for local adjustments. On the one hand their aim is to maximize the brake-traction overlap to enable the usage of regenerative energy on a fixed set of synchronized arrival and departure events. On the other hand, they include passenger related objectives such as the minimization of the generalized average travel time of all passengers and the minimization of the maximum increase in individual's generalized travel time. Wang et al. also provide a visualization of the Pareto frontier for these objectives on an instance of Dutch railways.

Our contribution

1. We propose a mixed integer programming (MIP) formulation for the problem of maximizing the brake-traction overlap (PESP-Energy), based on the Periodic Event Scheduling Problem (PESP) (Section 2.1) and including the decision which acceleration and braking processes are synchronized.
2. We extend this MIP formulation to the bicriteria problem that additionally aims at minimizing the passengers' travel time (Section 2.3) and run numerical experiments on a single transfer station. (Section 5)

For our theoretical investigation, we focus on the problem restricted to a single transfer station, for which we derive the following results:

3. We characterize the structure of optimal solutions for the two single-objective problems (Propositions 8 and 9 and Theorem 10).
4. We show that only minimizing the transfer times is already NP-hard for a single transfer station (Theorem 6).
5. Based on a special-form TSP, we obtain a polynomial-time algorithm with an additive performance guarantee (depending on the input parameters) for the energy objective (Theorem 16). We show for some special cases that its solution is optimal (Section 4.4).

2 Including the Brake-Traction Overlap in the Periodic Event Scheduling Problem

In the timetabling problem, we are given a set of lines $l \in \mathcal{L}$, which are given as sequences of served stations $v \in \mathcal{V}$. Every line will be served periodically with the given period of T.

2.1 PESP-Passenger – Minimizing the Travel Times

For the PESP model we are given bounds on the durations of activities (driving, waiting, transfers) as well as weights which correspond to the number of passengers performing each activity. The objective is to minimize the total travel time of all passengers. For this problem, the *event-activity-network* (EAN) $\mathcal{E} = (E, A)$ for given directed lines $\mathcal{L}$ serving stations $v \in \mathcal{V}$ is a directed graph on all arrival and departure events $E = E_{\text{arr}} \dot{\cup} E_{\text{dep}}$, given by

$$E_{\text{arr}} \coloneqq \big\{(v, \ell, \text{arr}) \mid \ell \in \mathcal{L} \text{ arrives at } v \in \mathcal{V}\big\},\ E_{\text{dep}} \coloneqq \big\{(v, \ell, \text{dep}) \mid \ell \in \mathcal{L} \text{ departs at } v \in \mathcal{V}\big\}.$$

The activities $A \coloneqq A_{\text{drive}} \mathbin{\dot\cup} A_{\text{wait}} \mathbin{\dot\cup} A_{\text{trans}}$ connect the events as follows:

$$\begin{aligned}
A_{\text{drive}} &\coloneqq \big\{((v_1,\ell,\text{dep}),(v_2,\ell,\text{arr})) \in E_{\text{dep}} \times E_{\text{arr}} \;\big|\; \ell \text{ serves } v_2 \text{ directly after } v_1\big\},\\
A_{\text{wait}} &\coloneqq \big\{((v,\ell,\text{arr}),(v,\ell,\text{dep})) \in E_{\text{arr}} \times E_{\text{dep}}\big\},\\
A_{\text{trans}} &\coloneqq \big\{((v,\ell_1,\text{arr}),(v,\ell_2,\text{dep})) \in E_{\text{arr}} \times E_{\text{dep}} \;\big|\; \ell_1 \neq \ell_2\big\}.
\end{aligned}$$

A timetable $\pi \colon E \to \{0,\dots,T-1\}$ assigns a time π_i to each event $i \in E$, meaning that the event takes place at all times from $\pi_i + T\mathbb{Z}$. We can associate the bounds on the driving, transfer and waiting times with the activities: For each activity $a \in A$ let $\Delta_a = [l_a, u_a]$ be the set of allowed durations with $l_a, u_a \in \mathbb{Z}$. Since we only determine the times modulo T, we can ignore multiples of T in the activity durations and therefore assume that $0 \leq l_a \leq T-1$ and $0 \leq u_a - l_a \leq T-1$. Then a timetable is *feasible* if the periodic tensions $x_{ij} \coloneqq (\pi_j - \pi_i - l_{ij}) \bmod T + l_{ij}$ lie within the provided bounds for all $ij \in A$. In this paper, we assume the bounds on the transfer arcs a we have $u_a = l_a + T - 1$ and, therefore, the bounds on the transfers do not impose feasibility constraints.

The classical PESP seeks to find a feasible schedule in this network. The PESP-Passenger problem aims to find one with minimal total travel time. Let $w(ij)$ be the total number of passengers performing the activity $ij \in A$. Then we minimize the weighted sum of the periodic tensions (cf. objective (1)) of all activities, yielding a timetable that minimizes the passengers' travel times. This leads to the following mixed integer linear program [7].

$$\begin{aligned}
&\text{(PESP-P)} \quad \min && \sum_{ij \in A} w(ij) x_{ij} && && (1)\\
&\text{subject to} && x_{ij} = \pi_j - \pi_i + p_{ij}T && \forall ij \in A && (2)\\
& && l_{ij} \leq x_{ij} \leq u_{ij} && \forall ij \in A && (3)\\
& && 0 \leq \pi_i \leq T-1 && \forall i \in E && (4)\\
& && x_{ij} \in \mathbb{R},\; p_{ij} \in \mathbb{Z} && \forall ij \in A && (5)\\
& && \pi_i \in \mathbb{Z} && \forall i \in E && (6)
\end{aligned}$$

The variables p_{ij} are called periodic offsets or modulo parameters and are chosen such that the periodic tensions x_{ij} lie within the bounds. This is ensured by constraints (2) and (3). Constraints (4) and (6) ensure that the timetable π takes only values within $\{0,\dots,T-1\}$.

2.2 PESP-Energy – Maximizing the Brake-Traction Overlap

Now we develop an extension of the PESP that allows to maximize our second objective function, the brake-traction overlap. In addition to the standard input, we are given the acceleration and braking times for all departures and arrivals, respectively. Our model, which we term *PESP-Energy*, is also based on an EAN $\mathcal{E} = (E, A)$. The events $E = E_{\text{arr}} \mathbin{\dot\cup} E_{\text{dep}}$ are derived from the set of stations $\mathcal{V}$ and the set of directed lines $\mathcal{L}$ as in PESP-Passenger. However, a different set of activities is considered. Specifically, we now have $A \coloneqq A_{\text{drive}} \mathbin{\dot\cup} A_{\text{wait}} \mathbin{\dot\cup} A_{\text{energy}}$ with A_{drive} and A_{wait} defined as above and

$$A_{\text{energy}} \coloneqq \big\{((v,\ell_1,\text{dep}),(v,\ell_2,\text{arr})) \in E_{\text{dep}} \times E_{\text{arr}}\big\}.$$

Such an energy arc is depicted in red in Figure 2a. The energy activities do not impose any constraints on the feasibility of a timetable, i.e., $\Delta_a = [0, T-1]$ for all $a \in A_{\text{energy}}$. For each arrival event $i \in E_{\text{arr}}$ the time t_i^{br} needed for braking, and the time t_j^{ac} needed for accelerating at each departure event $j \in E_{\text{dep}}$ are given. We assume that $t_j^{\text{ac}} + t_i^{\text{br}} < T$ for any $ji \in A_{\text{energy}}$.

We denote with $t_{ji}^{\min} := \min\{t_j^{\text{ac}}, t_i^{\text{br}}\}$ the minimum and with $t_{ji}^{\max} := \max\{t_j^{\text{ac}}, t_i^{\text{br}}\}$ the maximum of the acceleration and braking times associated with energy arc ji. We consider the periodic intervals of the acceleration and braking phases. By a periodic interval we mean

$$[a,b]_T := \begin{cases} [a \bmod T, b \bmod T] & \text{if } a \bmod T \leq b \bmod T, \\ [0, b \bmod T] \cup [a \bmod T, T) & \text{else.} \end{cases}$$

The length of a periodic interval is $\text{length}([a,b]_T) := (b-a) \bmod T$. The periodic interval of the acceleration phase after the departure event j is then $[\pi_j, \pi_j + t_j^{\text{ac}}]_T$ and, analogously, $[\pi_i - t_i^{\text{br}}, \pi_i]_T$ describes the braking phase before the arrival event i. The overlap of the two phases is then determined by the intersection of the periodic intervals. Note that due to the assumption that $t_j^{\text{ac}} + t_i^{\text{br}} < T$, this is again a periodic interval.

▶ **Definition 1** (Brake-Traction Overlap). *For $ji \in A_{\text{energy}}$ we define the brake-traction overlap resulting from a periodic timetable π as $o_{ji} := \text{length}\big([\pi_j, \pi_j + t_j^{\text{ac}}]_T \cap [\pi_i - t_i^{\text{br}}, \pi_i]_T\big)$.*

Clearly, the overlap does not depend on the exact times π_j and π_i but only on their difference, i.e., on the periodic tension x_{ji}. The following lemma gives a formula to compute it, using the function $\text{overlap}_a \colon [0,T) \to \mathbb{R}_{\geq 0}$ depicted in Figure 2b.

▶ **Lemma 2.** *For every $a \in A_{\text{energy}}$ with periodic tension x the brake-traction overlap is*

$$\text{overlap}_a(x) := \max\big\{\min\{x,\, t_a^{\min},\, t_a^{\max} + t_a^{\min} - x\},\ 0\big\}.$$

Proof. Let $a = ji$. There are two cases in which there is an empty intersection $[\pi_j, \pi_j + t_j^{\text{ac}}]_T \cap [\pi_i - t_i^{\text{br}}, \pi_i]_T$. First, the intersection is empty if $\pi_j \leq \pi_i$ and $\pi_j + t_j^{\text{ac}} < \pi_i - t_i^{\text{br}}$. This is the case whenever $t_j^{\text{ac}} + t_i^{\text{br}} < \pi_i - \pi_j = (\pi_i - \pi_j) \bmod T = x_{ji}$. The second case in which the intersection is empty is if $\pi_j > \pi_i$ and $\pi_j + t_j^{\text{ac}} < \pi_i + T - t_i^{\text{br}}$. This is true whenever $t_j^{\text{ac}} + t_i^{\text{br}} < \pi_i + T - \pi_j = (\pi_i - \pi_j) \bmod T = x_{ji}$. Hence, we have an empty intersection if and only if $t_j^{\text{ac}} + t_i^{\text{br}} - x_{ji} < 0$. In this case the overlap is $o_{ji} = 0$.

Provided that the intersection is non-empty, we receive the length of the overlap by the minimum of the lengths of the four intervals $[\pi_j, \pi_j + t_j^{\text{ac}}]_T$, $[\pi_i - t_i^{\text{br}}, \pi_i]_T$, $[\pi_j, \pi_i]_T$, $[\pi_i - t_i^{\text{br}}, \pi_j + t_j^{\text{ac}}]_T$. This yields

$$\begin{aligned} o_{ji} &= \min\{t_j^{\text{ac}}, t_i^{\text{br}}, (\pi_i - \pi_j) \bmod T, (t_j^{\text{ac}} + t_i^{\text{br}} - (\pi_i - \pi_j) \bmod T) \bmod T\} \\ &= \min\{t_j^{\text{ac}}, t_i^{\text{br}}, x_{ji}, (t_j^{\text{ac}} + t_i^{\text{br}} - x_{ji}) \bmod T\} \\ &= \min\{t_j^{\text{ac}}, t_i^{\text{br}}, x_{ji}, t_j^{\text{ac}} + t_i^{\text{br}} - x_{ji}\} \\ &= \min\{t_{ji}^{\min}, x_{ji}, t_{ji}^{\min} + t_{ji}^{\max} - x_{ji}\} \geq 0. \end{aligned}$$

The third equation holds by the assumption that we have a non-empty intersection. Therefore,

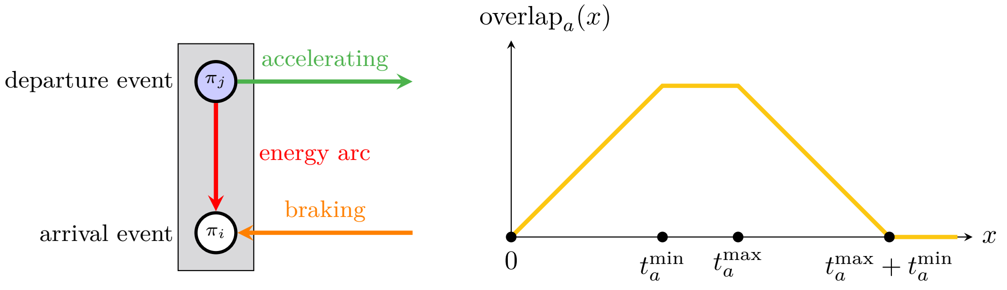
$$\min\{t_{ji}^{\min}, x_{ji}, t_{ji}^{\min} + t_{ji}^{\max} - x_{ji}\} \geq 0 \iff [\pi_j, \pi_j + t_j^{\text{ac}}]_T \cap [\pi_i - t_i^{\text{br}}, \pi_i]_T \neq \emptyset.$$

Hence, for the actual overlap of energy arc $a \in A_{\text{energy}}$ we obtain:

$$o_a = \text{overlap}_a(x) = \max\big\{\min\{x_a,\, t_a^{\min},\, t_a^{\max} + t_a^{\min} - x_a\},\ 0\big\}. \qquad ◀$$

The maximum possible overlap at $a \in A_{\text{energy}}$ is $o_a = t_a^{\min}$, which is achieved if and only if $t_a^{\min} \leq x_a \leq t_a^{\max}$. In this case, we say that there is *full overlap* on a.

Of course, the fact that energy can only be reused once must be taken into account in the model. Previous work [14] assumed a fixed matching between braking and accelerating trains. In contrast, we integrate these decisions directly into the model. Therefore, the



(a) Energy arc.

(b) Function $\mathrm{overlap}_a$ mapping tension x on a to brake-traction overlap.

Figure 2 Energy arc in EAN and brake-traction overlap as a function of the periodic tension.

problem PESP-Energy consists in finding a feasible periodic timetable π together with a matching $M \subset A_{\text{energy}}$ in $\mathcal{E}$ such that the sum of the brake-traction overlaps on the energy arcs in the matching is maximized (cf. (7)):

$$\begin{aligned}
&\text{(PESP-E)}\ \max && \sum_{ji \in A_{\text{energy}}} o_{ji} && && (7)\\
&\text{s.t.} && (2)\text{–}(4) && && \\
& && o_{ji} \le x_{ji} && \forall ji \in A_{\text{energy}} && (8)\\
& && o_{ji} \le t_{ji}^{\min} && \forall ji \in A_{\text{energy}} && (9)\\
& && o_{ji} \le t_{ji}^{\max} + t_{ji}^{\min} - x_{ji} + (1-\alpha_{ji})\Gamma && \forall ji \in A_{\text{energy}} && (10)\\
& && o_{ji} \le \alpha_{ji} \cdot \Gamma && \forall ji \in A_{\text{energy}} && (11)\\
& && \sum_{a \in A_{\text{energy}} \cap \delta^-(i)} \alpha_a \le 1 && \forall i \in E_{\text{arr}} && (12)\\
& && \sum_{a \in A_{\text{energy}} \cap \delta^+(j)} \alpha_a \le 1 && \forall j \in E_{\text{dep}} && (13)\\
& && o_{ji} \ge 0,\ \alpha_{ji} \in \{0,1\} && \forall ji \in A_{\text{energy}} && (14)\\
& && x_{ij} \ge 0,\ p_{ij} \in \mathbb{Z} && \forall ij \in A && (15)\\
& && \pi_i \in \mathbb{Z} && \forall i \in E && (16)
\end{aligned}$$

As we want to find a feasible timetable, the model also contains the constraints (2)–(4) from the standard PESP. The variable o_{ji} determines the brake-traction overlap and is bounded from above by the constraints (8)–(10) according to Lemma 2. The constant Γ is chosen large enough so that for $\alpha_{ji} \in \{0,1\}$ one of the constraints (10) and (11) does not impose a relevant bound on o_{ji}. It can be set to $\Gamma := \max\bigl\{\max\{t_{ji}^{\min}, T - (t_{ji}^{\max} + t_{ji}^{\min})\} \bigm| ji \in A_{\text{energy}}\bigr\}$. Constraints (12) and (13) ensure that the energy arcs chosen at each station form a matching. They set α_{ji} to 0 whenever the arc $ji \in A_{\text{energy}}$ is not chosen to be in the matching. Constraint (11) ensures that the overlap is not counted whenever $\alpha_{ji} = 0$.

We now compare the way to model the brake-traction overlap in (PESP-E) with the formulation of Wang et al. [14] for the timetable adjustment problem. For each energy arc, Wang et al. introduce two binary variables to decide whether there is a brake-traction overlap or not and thereby distinguish cases in which the periodic offset is 0 or 1. The next theorem formally states that the parts maximizing the brake-traction overlap are equivalent in both models. For a proof of this equivalence we refer to the appendix.

▶ **Theorem 3.** *The constraints (8)–(11) are equivalent to the constraints (18)–(26) in the appendix, taken from the model of Wang et al. [14], in the sense that for each energy arc* $a \in A_{\text{energy}}$ *and periodic timetable* π *with tension* x_a *the overlaps in the two models are equal.*

Next, we give an upper bound for the objective value of this maximization problem. To this end, we define weights for the energy arcs $ji \in A_{\text{energy}}$ as $w(ji) := t_{ji}^{\min}$.

▶ **Proposition 4.** *For an instance of PESP-Energy on the EAN* $\mathcal{E} = (E, A)$*, let* $S = (\pi, M)$ *be a feasible solution with objective value* $f(S)$*, and let* w^{opt} *be the maximum weight of a (perfect) matching in the graph* $G = (E, A_{\text{energy}})$ *with weights* $w(ji)$ *as defined above. Then* $f(S) \leq w^{\text{opt}}$*.*

Proof. Each overlap is bounded from above by both the corresponding acceleration and the corresponding braking time: $o_{ji} \leq t_j^{\text{ac}}$ and $o_{ji} \leq t_i^{\text{br}}$. ◀

2.3 The Bicriteria Problem

For real timetabling problems it is desirable to find timetables that enable the usage of regenerative energy as well as short travel times for the passengers. Hence, it is necessary to consider a bicriteria problem and study Pareto optimal solutions to find a good trade-off. The bicriteria MIP formulation consists of the objectives (1) and (7) under the constraints (2)–(14). However, solving only PESP-Passenger on large networks exactly is already computationally out of scope. To obtain a better understanding of the problem under the two objectives, we investigate the solution structures on a small network of one transfer station.

▶ **Definition 5** (One-Station Network). *An EAN* $\mathcal{E}_n = (E, A)$ *is called a* one-station network *with* n *lines if it is based on one station* $|\mathcal{V}| = 1$ *and* n *(directed) lines stopping at this station inducing the following events:*

$$E_{\text{arr}} := \big\{(\ell, \text{arr}) \bigm| \ell \in [n]\big\}, \qquad\qquad E_{\text{dep}} := \big\{(\ell, \text{dep}) \bigm| \ell \in [n]\big\}.$$

The activities $A = A_{\text{wait}} \mathbin{\dot\cup} A_{\text{trans}} \mathbin{\dot\cup} A_{\text{energy}}$ *connect the events as follows:*

$$\begin{aligned} A_{\text{wait}} &:= \big\{((\ell, \text{arr}), (\ell, \text{dep})) \in E_{\text{arr}} \times E_{\text{dep}}\big\}, \\ A_{\text{trans}} &:= \big\{((\ell_1, \text{arr}), (\ell_2, \text{dep})) \in E_{\text{arr}} \times E_{\text{dep}} \bigm| \ell_1 \neq \ell_2\big\}, \\ A_{\text{energy}} &:= \big\{((\ell_1, \text{dep}), (\ell_2, \text{arr})) \in E_{\text{dep}} \times E_{\text{arr}}\big\}. \end{aligned}$$

There are no driving activities in a one-station network. In the following, a one-station network $\mathcal{E}_n^{\text{pass}}$ for PESP-Passenger has the arcs $A_{\text{wait}} \cup A_{\text{trans}}$, while for PESP-Energy the arc set of the one-station network $\mathcal{E}_n^{\text{energy}}$ consists of $A_{\text{wait}} \cup A_{\text{energy}}$.

3 PESP-Passenger on a One-Station Network

The Periodic Event Scheduling Problem is NP-complete for any fixed $T \geq 3$, which can be proved by a reduction from the vertex colouring problem [9]. More recent work shows NP-hardness on a star network with turnaround loops [1]. Here, we show that the problem of finding a timetable minimizing the total transfer time on a *single* station is NP-hard as well.

▶ **Theorem 6.** *The problem PESP-Passenger is NP-hard even on a one-station network.*

Proof. We show NP-hardness by a reduction from the Max-Cut problem. Let I be an arbitrary instance of the Max-Cut problem, consisting of a graph $G = (V, R)$ and a weight function $w\colon R \to \mathbb{R}$. We search a bipartition (S, T) of the vertex set V such that the sum of the weights on the edges between the sets S and T, $\sum_{u \in S, v \in T} w(e_{uv})$, is maximal.

Based on I we define an instance I' of the PESP-Passenger problem on a one-station network: Let $\mathcal{E}_n^{\text{pass}} = (E_{\text{arr}} \,\dot\cup\, E_{\text{dep}}, A_{\text{wait}} \,\dot\cup\, A_{\text{trans}})$ be a one-station network with $n := |V|$ lines, inducing n arrival events and n departure events. Let the period time $T := 2$. The bounds on the waiting activities $ij \in A_{\text{wait}}$ are $l_{ij} = u_{ij} = 0$, and their weights are $w(ij) := 0$. For the transfer activities $ij \in A_{\text{trans}}$, let the bounds $l_{ij} = 1$ and $u_{ij} = 2$ and weights

$$w'(ij) := \begin{cases} w(\{i,j\}) & \text{if } \{i,j\} \in R, \\ 0 & \text{else.} \end{cases}$$

We want to show that any optimal solution to I' can be transformed to an optimal solution of I. Let π be an optimal timetable for I'. We define $S := \{i \in V \mid \pi_{(i,\text{dep})} = 0\}$ and $T := \{i \in V \mid \pi_{(i,\text{dep})} = 1\}$. To see that (S, T) is an optimal solution to the Max-Cut problem, i.e., that $\sum_{u \in S, v \in T} w(\{u, v\})$ is maximum, note that π minimizes the sum of the weights multiplied with the periodic tensions in I'. As every transfer arc has tension 1 or 2, this is the same as maximizing the sum of the weights of arcs with tension 1, which are exactly those between the sets S and T. Hence, (S, T) is a maximum-weight cut. ◀

Now we establish a special case in which we know the structure of an optimal solution.

▶ **Definition 7** (Basel Solution Structure). *A timetable π for a one-station network $\mathcal{E}_n^{\text{pass}}$ has a* Basel solution structure *if all arrival events are scheduled at the same time π^{arr} and all departure events at time π^{dep} such that $(\pi^{\text{dep}} - \pi^{\text{arr}}) \bmod T = l^{\max} := \max\{l_a \mid a \in A\}$.*

▶ **Proposition 8.** *Let $\mathcal{E}_n^{\text{pass}} = (E, A)$ be a one-station network with lower and upper bounds l_a, u_a on the arcs such that $u_a = l_a + T - 1$ for all transfer arcs $a \in A_{\text{trans}}$. Then any timetable π with the Basel solution structure minimizes the total travel time independently of the weights if and only if $l_a = l_{a'}$ for all $a, a' \in A_{\text{trans}} \cup A_{\text{wait}}$.*

Proof. First, let us assume that $l_a = l_{a'}$ for all $a, a' \in A_{\text{trans}} \cup A_{\text{wait}}$. Let π be a timetable with the Basel solution structure. Then the periodic tensions induced by π are $x_{ij} = (\pi_j - \pi_i - l_{ij}) \bmod T + l_{ij} = (l^{\max} - l^{\max}) \bmod T + l^{\max} = l^{\max}$ for all $ij \in A_{\text{trans}} \cup A_{\text{wait}}$. As we cannot do better than attaining the lower bounds on the tensions, π must be optimal.

Let us now assume that there is an arc $a' \in A_{\text{trans}} \cup A_{\text{wait}}$ with $l_{a'} < l^{\max}$. In the following we find a weight vector w for which π is not optimal. Let $a' = i'j'$. Then the following timetable π' achieves a lower objective value than π for the following weight vector w:

$$w(ij) := \begin{cases} 1 & \text{if } ij = i'j', \\ 0 & \text{else,} \end{cases} \qquad \pi'_i := \begin{cases} 0 & \text{if } i = i', \\ l_{i'j'} & \text{if } i = j', \\ \text{arbitrary feasible values} & \text{else.} \end{cases}$$

This is possible as only the waiting activities impose feasibility constraints. The weighted sum of the periodic tensions w.r.t. π is $l^{\max}$, and it is $l_{a'}$ w.r.t. π'. By assumption, $l_{a'} < l^{\max}$, hence, π is not optimal. ◀

4 PESP-Energy on a One-Station Network

4.1 The Timetable for a Given Matching

An EAN for this problem is a bipartite graph with partition classes E_{arr} and E_{dep}. In a one-station network, the set of waiting activities constitutes a perfect matching from E_{arr} to E_{dep}. These activities impose the only feasibility constraints on the timetable π. In contrast, the energy activities solely influence the objective value. Hence, arrival and departure times of different lines are not restricted by any PESP constraint. For any matching $M \subset A_{\text{energy}} = E_{\text{dep}} \times E_{\text{arr}}$, the graph $(E, A_{\text{wait}} \dot\cup M)$ is a union of node-disjoint directed cycles and directed paths. A timetable with maximum brake-traction overlap on the matching arcs can be determined for each connected component of this graph separately. The following proposition describes the structure of an optimal timetable for a directed cycle.

▶ **Proposition 9.** *Let $\mathcal{E}_n^{\text{energy}}$ be a one-station network, let $M \subset A_{\text{energy}}$ be a matching, and let $C \subset A_{\text{wait}} \dot\cup M$ be a directed cycle. We write the cycle as $C = a_1, b_1, a_2, b_2 \ldots, a_m, b_m$ with $a_j \in M$, $b_j \in A_{\text{wait}}$, and $t^{\min}_{a_j} \geq t^{\min}_{a_1}$ for all $j \in [m]$. There is an optimal timetable π for PESP-Energy restricted to C such that we have full overlap $o_{a_j} = t^{\min}_{a_j}$ for all $j \in \{2, \ldots, m\}$.*

Proof. Let π be a timetable maximizing the brake-traction overlap on the energy arcs of C. Among all such timetables, we consider one with the maximum number of arcs a_j with $j \in \{2, \ldots, m\}$ having full overlap. Assume that some arc a_j with $j \in \{2, \ldots, m\}$ does not have full overlap. To derive a contradiction, we modify the timetable π on C so that a_j has full overlap, while preserving full overlap on all other arcs and not reducing the total overlap.

To this end, we first define the new tensions x' and then construct a timetable π' inducing them. Let x be the tension induced by π. For $k \in [m]$ we set $x'_{b_k} := x_{b_k}$ and

$$x'_{a_k} := \begin{cases} x_{a_k} & \text{for } k \neq \{1, j\}, \\ (x_{a_1} - \delta) \bmod T & \text{for } k = 1, \\ x_{a_j} + \delta & \text{for } k = j, \end{cases} \qquad \text{where} \quad \delta := \begin{cases} t^{\min}_{a_j} - x_{a_j} & \text{if } x_{a_j} < t^{\min}_{a_j}, \\ t^{\max}_{a_j} - x_{a_j} & \text{if } x_{a_j} > t^{\max}_{a_j}. \end{cases}$$

Note that this covers all cases because for $t^{\min}_{a_j} \leq x_{a_j} \leq t^{\max}_{a_j}$ the activity a_j would have full overlap, contradicting our assumption. We define the periodic timetable π' as follows: We enumerate the nodes of the cycle so that $a_k = (2k-1, 2k)$ for $k \in [m]$ and $b_k = (2k, 2k+1)$ for $k \in [m-1]$. The nodes with even number correspond to arrivals and with odd number to departures. We set $\pi'_1 := 0$, $\pi'_{2k} := (\pi'_{2k-1} + x'_{a_k}) \bmod T$ for $k \in [m]$, and $\pi'_{2k+1} := (\pi'_{2k} + x'_{b_k}) \bmod T$ for $k \in [m-1]$. This adheres to the prescribed tensions on all arcs a_k, $k \in [m]$, and b_k, $k \in [m-1]$. The tension on b_m is congruent to $\pi'_1 - \pi'_{2m} = -\pi'_{2m} = -\sum_{k=1}^{m} x'_{a_k} - \sum_{k=1}^{m-1} x'_{b_k} \equiv -\sum_{k=1}^{m} x_{a_k} - \sum_{k=1}^{m-1} x_{b_k} \equiv x_{b_m} \pmod{T}$. The last congruence holds since the periodic tension x sums up to a multiple of T due to the cycle periodicity.

For $a \in M$ let $o'_a := \text{overlap}_a(x'_a)$. The only matching arcs whose tensions have changed are a_1 and a_j. We have $x'_{a_j} \in \{t^{\min}_{a_j}, t^{\max}_{a_j}\}$, and thus a_j has now full overlap, i.e., $o'_{a_j} = t^{\min}_{a_j}$. It holds that

$$\begin{aligned} o'_{a_j} - o_{a_j} = t^{\min}_{a_j} - o_{a_j} &= \begin{cases} t^{\min}_{a_j} - 0 & \text{if } x_{a_j} > t^{\min}_{a_j} + t^{\max}_{a_j}, \\ t^{\min}_{a_j} - x_{a_j} & \text{if } x_{a_j} < t^{\min}_{a_j}, \\ t^{\min}_{a_j} - (t^{\max}_{a_j} + t^{\min}_{a_j} - x_{a_j}) & \text{if } t^{\max}_{a_j} < x_{a_j} \leq t^{\max}_{a_j} + t^{\min}_{a_j}, \end{cases} \\ &= \begin{cases} t^{\min}_{a_j} & \text{if } x_{a_j} > t^{\min}_{a_j} + t^{\max}_{a_j}, \\ |\delta| & \text{else}, \end{cases} \\ &\geq \min\{|\delta|, t^{\min}_{a_j}\}. \end{aligned}$$

Similarly, $o_{a_1} - o'_{a_1} \leq \min\{|\delta|, t^{\min}_{a_1}\} \leq \min\{|\delta|, t^{\min}_{a_j}\}$, hence the decrease of the overlap on a_1 is at most the increase of the overlap on a_j. Since no other overlaps have changed, the sum of all overlaps cannot have decreased, i.e., we have found a solution whose objective is not worse but which has more arcs a_j with $j \in \{2, \dots, m\}$ with full overlap. ◀

We can also regard a connected component being a directed path as a cycle whose missing edge has overlap 0. Hence, by Proposition 9, there is an optimal timetable for this component such that all energy edges in the path have full overlap. While the proposition describes the structure of an optimal timetable for each connected component resulting from a fixed matching M, we are interested in a global optimum, comprising the matching. Hence, we need to investigate the structure of an optimal matching.

4.2 The Matching of Energy Arcs

The following result bounds the number of arcs in the matching of a globally optimal solution.

▶ **Theorem 10.** *In every optimal solution $S = (\pi, M)$ to PESP-Energy on a one-station network $\mathcal{E}^{\text{energy}}_n$ the matching M contains at least $n-1$ arcs.*

Proof. Let $S = (\pi, M)$ be an optimal solution to PESP-Energy with $|M| < n-1$, so at least two connected components of $(E, A_{\text{wait}} \dot\cup M)$ are directed paths P_1, P_2. For $k \in \{1, 2\}$ let $i_k \in E_{\text{arr}}$ be the start and $j_k \in E_{\text{dep}}$ be the end node of P_k.

Let $c := \pi_{j_1} + t^{\min}_{j_1 i_2} - \pi_{i_2}$, and let us define the following timetable

$$\pi'_v := \begin{cases} \pi_v & \text{if } v \in E \setminus V(P_2), \\ (\pi_v + c) \bmod T & \text{if } v \in V(P_2). \end{cases}$$

Now, let $S' = (\pi', M')$ with $M' = M \cup \{j_1 i_2\}$. For the tensions on M', we obtain:

$$x'_{ji} = \begin{cases} (\pi'_i - \pi'_j) \bmod T = (\pi_i - \pi_j) \bmod T = x_{ji} & \text{for } ji \in M, \\ (\pi'_{i_2} - \pi'_{j_1}) \bmod T = t^{\min}_{j_1 i_2} & \text{for } j = j_1, i = i_2. \end{cases}$$

Hence, we obtain the brake-traction overlaps

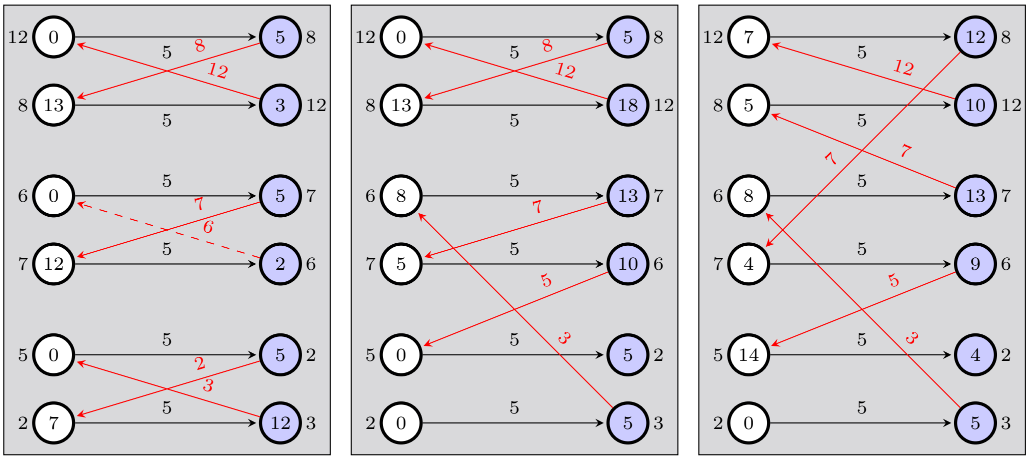
$$o'_{ji} = \begin{cases} o_{ji} & \text{if } ji \in M, \\ t^{\min}_{j_1 i_2} & \text{if } ji = j_1 i_2. \end{cases}$$

Consequently, S' yields a better objective value than S, so S cannot be optimal. ◀

▶ **Corollary 11.** *There is a unique perfect matching M^p which is obtained by extending the matching M of an optimal solution to PESP-Energy on a one-station network.*

This yields another way of looking at an optimal solution to PESP-Energy on a one-station network. A perfect matching $M^p \subset A_{\text{energy}}$ corresponds one-to-one to a permutation $\varphi\colon [n] \to [n]$ of the trains (lines) in a one-station network. This is given by $\varphi(\ell) = k$ if and only if $((\ell, \text{dep}), (k, \text{arr})) \in M^p$. The directed cycles in $M^p \cup A_{\text{wait}}$ then correspond to the cycles of the permutation.

Recall that we can find an upper bound for the objective value of a PESP-Energy instance by calculating the maximum-weight perfect matching on the energy arcs $a \in A_{\text{energy}}$ w.r.t. the weights $w\colon A_{\text{energy}} \to \mathbb{R}$ with $w(a) = t^{\min}_a$ (cf. Proposition 4). This matching can be found easily by a greedy algorithm for the weights in our problem. Sorting both t^{br}_i, $i \in E_{\text{arr}}$, and t^{ac}_j, $j \in E_{\text{dep}}$, according to their sizes, we obtain the permutations ρ and σ with $t^{\text{br}}_{\rho(1)} \leq \dots \leq t^{\text{br}}_{\rho(n)}$ and $t^{\text{ac}}_{\sigma(1)} \leq \dots \leq t^{\text{ac}}_{\sigma(n)}$. Then $M_{\text{greedy}} := \big\{((\sigma(i), \text{dep}), (\rho(i), \text{arr})) \in E_{\text{dep}} \times E_{\text{arr}} \bigm| i \in [n]\big\}$ is a perfect matching with maximum weight in the graph (E, A_{energy}).



(a) Greedy matching, weight: 38, total overlap: 32.

(b) Optimal solution, total overlap: 35.

(c) Hamiltonian path, total overlap: 34.

Figure 3 Example of non-optimal greedy and Hamiltonian path matchings for $T = 15$. The timetable π is written in the nodes. To the left of the arrival nodes (white) the braking times are given, and to the right of the departure nodes (blue) there are the acceleration times. The numbers on the waiting (black) and energy (red) arcs correspond to the periodic tensions. For the full red arcs, they also correspond to the achieved overlap.

4.3 A Hamiltonian Path Algorithm/Heuristic

A lower bound on the optimal objective value of PESP-Energy can be obtained by a maximum-weight Hamiltonian path on $\mathcal{E}_n^{\text{energy}}$ with respect to the weights $w\colon A \to \mathbb{R}$ defined by $w(ji) := t_{ji}^{\min}$ for $ji \in A_{\text{energy}}$ and $w(ij) := \Gamma$ for $ij \in A_{\text{wait}}$, where Γ is a big number. The choice of weights ensures that the path consists of n waiting arcs and $n-1$ energy arcs. Adding the arc from the end node of the path to its start node creates a cycle. Due to Proposition 9, we know that in that cycle, we can obtain full overlap on the best $n-1$ energy arcs, so we obtain at least overlap equal to the weight of the path's energy activities. Hence, this yields a lower bound on the optimal overlap achievable. In Figure 3 we can see that neither weight of the greedy matching is always obtained as overlap nor does a maximum-weight Hamiltonian path necessarily yield an optimal solution. In Figure 3a the greedy matching together with the waiting activity $M \cup A_{\text{wait}}$ decomposes into three cycles. Due to the cycle periodicity, however, we cannot obtain full overlap in the second cycle. There is no overlap on the dashed arc. While the greedy matching has weight 38, only an overlap of 32 can be obtained from the matching. In Figure 3b an optimal solution is depicted. We can see a decomposition of one cycle and one path, which cannot be closed due to cycle periodicity. The achieved overlap is 35. In Figure 3c, we can see a maximum-weight Hamiltonian path with full overlap on all energy arcs. In total an overlap of 34 is achieved. Due to the cycle periodicity it is not possible to obtain overlap on the missing energy arc. We show now that this lower bound can be computed in polynomial time.

▶ **Theorem 12.** *A maximum-weight Hamiltonian path on a one-station network $\mathcal{E}_n^{\text{energy}}$ with weights w can be found in polynomial time.*

In order to prove this theorem, we show that PESP-Energy on a one-station network can be transformed to the problem of sequencing a machine with variable state, for which a polynomial-time algorithm is known, see [4]. To simplify the notation, in this section we write

$t_\ell^{\mathrm{ac}} := t_{(\ell,\mathrm{dep})}^{\mathrm{ac}}$ and $t_\ell^{\mathrm{br}} := t_{(\ell,\mathrm{arr})}^{\mathrm{br}}$ for $\ell \in [n]$. We consider the complete directed graph $G = ([n], \mathcal{A})$ on the set of all lines and weights $w\colon \mathcal{A} \to \mathbb{R}$ defined by $w_G(k\ell) := \min\{t_k^{\mathrm{ac}}, t_\ell^{\mathrm{br}}\}$. We need the following three lemmas.

▶ **Lemma 13.** *If P_G is a maximum-weight Hamiltonian path in G w.r.t. w_G, then $P := \{((k,\mathrm{dep}),(\ell,\mathrm{arr}) \mid k\ell \in P_G\} \cup \{((\ell,\mathrm{arr}),(\ell,\mathrm{dep})) \mid \ell \in [n]\}$ is a maximum-weight Hamiltonian path in $\mathcal{E}_n^{\mathrm{energy}}$ w.r.t. w.*

Proof. First, P is the arc set of a Hamiltonian path. It has weight $w(P) = w(P_G) + n\Gamma$. Let P^{opt} be a maximum-weight Hamiltonian path in $\mathcal{E}_n^{\mathrm{energy}}$. By the large choice of Γ, this must contain n waiting arcs and hence contains exactly $n-1$ energy arcs $((k,\mathrm{dep}),(\ell,\mathrm{arr}))$. Then $(P^{\mathrm{opt}})_G := \{(k,\ell) \mid ((k,\mathrm{dep}),(\ell,\mathrm{arr})) \in P^{\mathrm{opt}}\}$ is a Hamiltonian path in G of weight $w_G((P^{\mathrm{opt}})_G) = w(P^{\mathrm{opt}}) - n\Gamma$. Therefore, $w(P) = w(P_G) + n\Gamma \geq w((P^{\mathrm{opt}})_G) + n\Gamma = w(P^{\mathrm{opt}})$ and P is a maximum-weight Hamiltonian path in $\mathcal{E}_n^{\mathrm{energy}}$. ◀

▶ **Lemma 14.** *Any maximum-weight Hamiltonian cycle in G w.r.t. w_G contains a maximum-weight Hamiltonian path. Conversely, any maximum-weight Hamiltonian path can be closed to a maximum-weight Hamiltonian cycle.*

Proof. Let $k := \arg\min\{t_{(\ell,\mathrm{dep})}^{\mathrm{ac}}, t_\ell^{\mathrm{br}} \mid \ell \in [n]\}$. W.l.o.g. let us assume $t_{(k,\mathrm{arr})}^{\mathrm{br}} \leq t_k^{\mathrm{ac}}$. We know that all incoming arcs of k have weight $w_G^{\min} := t_{(k,\mathrm{arr})}^{\mathrm{br}}$. Let $\mathcal{C}^{\mathrm{opt}}$ be a maximum-weight Hamiltonian cycle. Since this must visit k, it must contain an arc a of weight $w_G^{\min}$. Then $\mathcal{P} := \mathcal{T}^{\mathrm{opt}} \setminus \{a\}$ is a Hamiltonian path with weight $w(\mathcal{P}) = w_G(\mathcal{C}^{\mathrm{opt}}) - w_G^{\min}$.

Let now $\mathcal{P}^{\mathrm{opt}}$ be a maximum-weight Hamiltonian path, and let v be the first and u be the last vertex in $\mathcal{P}^{\mathrm{opt}}$. Then $\mathcal{C} := \mathcal{P}^{\mathrm{opt}} \cup \{uv\}$ is a Hamiltonian tour with weight $w_G(\mathcal{C}) = w_G(\mathcal{P}^{\mathrm{opt}}) + w_G(uv) \geq w_G(\mathcal{P}^{\mathrm{opt}}) + w_G^{\min}$.

Together, both $\mathcal{P}$ and $\mathcal{C}$ must be optimal since $w_G(\mathcal{P}) = w_G(\mathcal{C}^{\mathrm{opt}}) - w_G^{\min} \geq w_G(\mathcal{C}) - w_G^{\min} \geq w_G(\mathcal{P}^{\mathrm{opt}})$ and $w_G(\mathcal{C}) \geq w_G(\mathcal{P}^{\mathrm{opt}}) + w_G^{\min} \geq w_G(\mathcal{P}) + w_G^{\min} = w(\mathcal{C}^{\mathrm{opt}})$. ◀

▶ **Lemma 15.** *Let $C_1, C_2 \subset \mathcal{A}$ be two Hamiltonian cycles in G. Consider a second weight function $w'\colon \mathcal{A} \to \mathbb{R}$ defined by $w'(k\ell) := |t_k^{\mathrm{ac}} - t_\ell^{\mathrm{br}}|$. If $w(C_1) \leq w(C_2)$, then $w'(C_1) \geq w'(C_2)$. Hence, a maximum-weight Hamiltonian cycle w.r.t. w is a minimum-weight Hamiltonian cycle w.r.t. w'.*

Proof. For the weight w' of a Hamiltonian cycle C we get

$$\begin{aligned} w'(C) &= \sum_{k\ell \in C} |t_k^{\mathrm{ac}} - t_\ell^{\mathrm{br}}| = \sum_{k\ell \in C} \left(\max\{t_k^{\mathrm{ac}}, t_\ell^{\mathrm{br}}\} - \min\{t_k^{\mathrm{ac}}, t_\ell^{\mathrm{br}}\}\right) \\ &= \sum_{k\ell \in C} \left(\max\{t_k^{\mathrm{ac}}, t_\ell^{\mathrm{br}}\} + \min\{t_k^{\mathrm{ac}}, t_\ell^{\mathrm{br}}\} - 2 \cdot \min\{t_k^{\mathrm{ac}}, t_\ell^{\mathrm{br}}\}\right) \\ &= \sum_{k\ell \in C} \left(t_k^{\mathrm{ac}} + t_\ell^{\mathrm{br}} - 2 \cdot \min\{t_k^{\mathrm{ac}}, t_\ell^{\mathrm{br}}\}\right) = \sum_{k \in [n]} (t_k^{\mathrm{ac}} + t_k^{\mathrm{br}}) - 2 \cdot w_G(C), \end{aligned}$$

where the first summand in the last expression is constant. Hence, if $w(C_1) \leq w(C_2)$, then $w'(C_1) \geq w'(C_2)$. ◀

Now, we can prove Theorem 12.

Proof of Theorem 12. The problem of sequencing a one state-variable machine from [4] is defined as follows. We consider N jobs $J_1, J_2, \ldots, J_N$ which are to be done on one machine in some order. For each job J_i we know the required starting state of the machine represented by the real number A_i and the machine's state after the completion of job J_i represented by the real number B_i. When job J_l follows job J_k, we need to change the machine's state from B_k to A_l. The cost c_{kl} of this change is defined as

$$c_{kl} := \begin{cases} \int\limits_{B_k}^{A_l} f(x)dx & \text{if } A_l \geq B_k, \\ \int\limits_{A_l}^{B_k} g(x)dx & \text{if } A_l < B_k. \end{cases}$$

Here, f and g are integrable functions such that $f(x) + g(x) \geq 0$ for all $x \in \mathbb{R}$. The problem is to find a sequence of jobs such that the sum of the costs for changing the state of the machine between consecutive jobs is minimized. The polynomial-time algorithm developed in [4] requires the prescription of an initial state B_{N+1} and a final state A_{N+1} of the machine so that it becomes the problem of finding a tour $J_{N+1}J_{i_1} \dots J_{i_N}J_{N+1}$ with the artificial job J_{N+1} minimizing the total state transition cost.

Now, consider finding a maximum-weight Hamiltonian path in $\mathcal{E}_n^{\text{energy}}$, w.r.t. the weights w. By Lemma 13, this is equivalent to finding a maximum-weight Hamiltonian path in G w.r.t. w_G. By means of Lemma 14, this can be reduced to finding a Hamiltonian cycle in G which corresponds to finding a minimum-weight Hamiltonian cycle in G w.r.t. the weights w' defined in Lemma 15. We reduce this problem to solving the sequencing problem of finding a closed tour on a set of jobs on the following instance I^{seq}.

For every directed line $\ell \in [n]$ we define a job ℓ with $A_\ell := t_\ell^{\text{br}}$ and $B_\ell := t_\ell^{\text{ac}}$. The functions for the state transition costs are defined as $f(x) = g(x) := 1$ for all $x \in \mathbb{R}$. This yields the costs $c_{kl} = A_l - B_k$ if $A_l \geq B_k$ and $c_{kl} = B_k - A_l$ if $A_l < B_k$. In other words, $c_{kl} = |A_l - B_k| = w'(k\ell)$. Therefore, the cost of any cyclic tour of the jobs equals the weight of the corresponding Hamiltonian tour in G w.r.t. w'. ◀

The maximum-weight Hamiltonian path yields a feasible solution to PESP-Energy on a one-station network. We can guarantee that the objective value of this solution is not further away from the optimal objective value than the smallest of the largest acceleration and the largest braking time. This follows from the following theorem, which bounds the difference between the lower bound and the upper bound from the greedy matching of Section 4.2.

▶ **Theorem 16.** *Let H be a Hamiltonian path in $\mathcal{E}_n^{\text{energy}}$ of maximum weight. Then it holds*

$$w(M_{\text{greedy}}) - w(H \cap A_{\text{energy}}) \leq \min\bigl\{\max\{t_\ell^{\text{br}} \mid \ell \in [n]\},\ \max\{t_\ell^{\text{ac}} \mid \ell \in [n]\}\bigr\}.$$

Proof. We iteratively convert the greedy matching M_{greedy}^0 into a matching inducing a Hamiltonian cycle and bound the total reduction of weight in this process. Finally, we delete one edge to obtain a Hamiltonian path.

Let us assume that $t_1^{\text{ac}} \leq \dots \leq t_n^{\text{ac}}$ holds, and let φ^0 denote the permutation obtained by M_{greedy}^0 such that $t_{\varphi^0(1)}^{\text{br}} \leq \dots \leq t_{\varphi^0(n)}^{\text{br}}$. Then $M_{\text{greedy}}^0 = \{((\ell, \text{dep}), (\varphi^0(\ell), \text{arr})) \mid \ell \in [n]\}$. The permutation φ^i corresponds to the perfect matching M^i obtained in iteration i.

In each iteration, we obtain the matching M^i as follows from the matching M^{i-1}. Let $C \subseteq M^{i-1} \cup A_{\text{wait}}$ be the cycle containing $(1, \text{dep})$. If C is a Hamiltonian cycle, we are done. Otherwise, there is a smallest ℓ such that $(\ell, \text{dep}) \in C$ but $(\ell + 1, \text{dep}) \notin C$. We define the new permutation φ^i as follows:

$$\varphi^i(x) := \begin{cases} \varphi^{i-1}(\ell+1) & \text{if } x = \ell, \\ \varphi^{i-1}(\ell) & \text{if } x = \ell + 1, \\ \varphi^{i-1}(x) & \text{else.} \end{cases}$$

For the new matching M^i we have, $M^i = M^{i-1} \cup \{((\ell, \mathrm{dep}), (\varphi^{i-1}(\ell+1), \mathrm{arr})), ((\ell+1, \mathrm{dep}), (\varphi^{i-1}(\ell), \mathrm{arr}))\} \setminus \{((\ell, \mathrm{dep}), (\varphi^{i-1}(\ell), \mathrm{arr})), ((\ell{+}1, \mathrm{dep}), (\varphi^{i-1}(\ell{+}1), \mathrm{arr}))\}$. The cycle's length increases by this operation and for the weight of the new matching M^i, we get

$$\begin{aligned} w(M^i) = w(M^{i-1}) &+ \min\{t^{\mathrm{ac}}_{\ell}, t^{\mathrm{br}}_{\varphi^{i-1}(\ell+1)}\} + \min\{t^{\mathrm{ac}}_{\ell+1}, t^{\mathrm{br}}_{\varphi^{i-1}(\ell)}\} \\ &- \min\{t^{\mathrm{ac}}_{\ell}, t^{\mathrm{br}}_{\varphi^{i-1}(\ell)}\} - \min\{t^{\mathrm{ac}}_{\ell+1}, t^{\mathrm{br}}_{\varphi^{i-1}(\ell+1)}\} \\ = w(M^{i-1}) &- |[t^{\mathrm{ac}}_{\ell}, t^{\mathrm{ac}}_{\ell+1}] \cap [t^{\mathrm{br}}_{\varphi^{i-1}(\ell)}, t^{\mathrm{br}}_{\varphi^{i-1}(\ell+1)}]|. \end{aligned}$$

Further, we know that $[t^{\mathrm{br}}_{\varphi^{i-1}(\ell)}, t^{\mathrm{br}}_{\varphi^{i-1}(\ell+1)}] \subset [t^{\mathrm{br}}_{\varphi^0(1)}, t^{\mathrm{br}}_{\varphi^0(n)}]$. Hence, the length of the intersection can be bounded by

$$|[t^{\mathrm{ac}}_{\ell}, t^{\mathrm{ac}}_{\ell+1}] \cap [t^{\mathrm{br}}_{\varphi^{i-1}(\ell)}, t^{\mathrm{br}}_{\varphi^{i-1}(\ell+1)}]| \leq |[t^{\mathrm{ac}}_{\ell}, t^{\mathrm{ac}}_{\ell+1}] \cap [t^{\mathrm{br}}_{\varphi^0(1)}, t^{\mathrm{br}}_{\varphi^0(n)}]|$$

and, therefore $w(M^i) \geq w(M^{i-1}) - |[t^{\mathrm{ac}}_{\ell}, t^{\mathrm{ac}}_{\ell+1}] \cap [t^{\mathrm{br}}_{\varphi^0(1)}, t^{\mathrm{br}}_{\varphi^0(n)}]|$.

Let k be the number of iterations until we obtain a matching M^k such that $M^k \cup A_{\mathrm{wait}}$ corresponds to a Hamiltonian cycle. In the following ℓ_i denotes the smallest $\ell \in [n-1]$ such that $(\ell, \mathrm{dep}) \in C$ and $(\ell+1, \mathrm{dep}) \notin C$ in iteration i. Further, it holds $k \leq n-1$ as there are n different trains. Hence, we can overestimate the sum as follows:

$$\sum_{i=1}^{k} |[t^{\mathrm{ac}}_{\ell_i}, t^{\mathrm{ac}}_{\ell_i+1}] \cap [t^{\mathrm{br}}_{\varphi^0(1)}, t^{\mathrm{br}}_{\varphi^0(\ell_i+1)}]| \leq \sum_{\ell=1}^{n-1} |[t^{\mathrm{ac}}_{\ell}, t^{\mathrm{ac}}_{\ell+1}] \cap [t^{\mathrm{br}}_{\varphi^0(1)}, t^{\mathrm{br}}_{\varphi^0(n)}]|$$

Instead of just summing up the length of the intervals for the corresponding train ℓ in each iteration, we sum over the lengths of all possible choices for ℓ. We get the following bound:

$$\begin{aligned} w(M^k) &\geq w(M^0_{\mathrm{greedy}}) - \sum_{i=1}^{k} |[t^{\mathrm{ac}}_{\ell_i}, t^{\mathrm{ac}}_{\ell_i+1}] \cap [t^{\mathrm{br}}_{\varphi^0(1)}, t^{\mathrm{br}}_{\varphi^0(\ell_i+1)}| \\ &\geq w(M^0_{\mathrm{greedy}}) - \sum_{\ell=1}^{n-1} |[t^{\mathrm{ac}}_{\ell}, t^{\mathrm{ac}}_{\ell+1}] \cap [t^{\mathrm{br}}_{\varphi^0(1)}, t^{\mathrm{br}}_{\varphi^0(n)}]|. \end{aligned}$$

As the intervals $[t^{\mathrm{ac}}_{\ell}, t^{\mathrm{ac}}_{\ell+1}]$ intersect only in one point (of length 0), we can bound the sum of the intersections as follows:

$$\begin{aligned} &\sum_{\ell=1}^{n-1} |[t^{\mathrm{ac}}_{\ell}, t^{\mathrm{ac}}_{\ell+1}] \cap [t^{\mathrm{br}}_{\varphi^0(1)}, t^{\mathrm{br}}_{\varphi^0(n)}]| \leq \left| \bigcup_{\ell=1}^{n-1} [t^{\mathrm{ac}}_{\ell}, t^{\mathrm{ac}}_{\ell+1}] \right| = t^{\mathrm{ac}}_{n} - t^{\mathrm{ac}}_{1}, \\ &\sum_{\ell=1}^{n-1} |[t^{\mathrm{ac}}_{\ell}, t^{\mathrm{ac}}_{\ell+1}] \cap [t^{\mathrm{br}}_{\varphi^0(1)}, t^{\mathrm{br}}_{\varphi^0(n)}]| \leq |[t^{\mathrm{br}}_{\varphi^0(1)}, t^{\mathrm{br}}_{\varphi^0(n)}]| = t^{\mathrm{br}}_{\varphi^0(n)} - t^{\mathrm{br}}_{\varphi^0(1)}. \end{aligned}$$

Thus, $w(M^k) \geq w(M^0_{\mathrm{greedy}}) - \min\{t^{\mathrm{ac}}_{n} - t^{\mathrm{ac}}_{1}, t^{\mathrm{br}}_{\varphi^0(n)} - t^{\mathrm{br}}_{\varphi^0(1)}\}$. In order to receive a Hamiltonian path $H \subseteq M^k \cup A_{\mathrm{wait}}$, we delete one edge from the matching M^k. As we want to maximize the path's weight, we choose the edge with the lowest weight. Due to the weight structure, this weight is $\min\{t^{\mathrm{ac}}_{1}, t^{\mathrm{br}}_{\varphi^0(1)}\}$. For the difference of the weights of the greedy matching M_{greedy} and the weight of the energy arcs in H, we get:

$$\begin{aligned} w(M_{\mathrm{greedy}}) - w(H \cap A_{\mathrm{energy}}) &\leq \min\{t^{\mathrm{ac}}_{n} - t^{\mathrm{ac}}_{1}, t^{\mathrm{br}}_{\varphi^0(n)} - t^{\mathrm{br}}_{\varphi^0(1)}\} + \min\{t^{\mathrm{ac}}_{1}, t^{\mathrm{br}}_{\varphi^0(1)}\} \\ &\leq \min\{t^{\mathrm{ac}}_{n}, t^{\mathrm{br}}_{\varphi^0(n)}\}. \end{aligned}$$

The weight of the path H is a lower bound for the weight of an optimal Hamiltonian path. ◀

4.4 Two Special Cases Solvable in Polynomial Time

There are some special cases in which we can solve PESP-Energy on a one-station network in polynomial time. In the first case, all braking times and all acceleration times are equal.

▶ **Proposition 17.** *Let I be an instance of PESP-Energy on a one-station network with n lines such that all acceleration times are equal and all braking times are equal, i.e., $t^{\mathrm{ac}} = t_j^{\mathrm{ac}}$ and $t^{\mathrm{br}} = t_i^{\mathrm{br}}$ for all $i, j \in [n]$. Then, there is an optimal solution to I consisting of one cycle of all lines in arbitrary order. This can be found in polynomial time.*

Proof. Assume that there is no optimal solution consisting of a single cycle, and consider an optimal solution (M, π) with the minimum number of cycles. Let C_1, C_2 be two different cycles in $M \cup A_{\mathrm{wait}}$, and let $a_k = (j_k, i_k) \in A_{\mathrm{energy}} \cap C_k$ for $k = 1, 2$. Consider the alternative solution with $M' := (M \setminus \{a_1, a_2\}) \cup \{j_1 i_2, j_2 i_1\}$. Then M' induces a big cycle on the node set $V(C_1) \cup V(C_2)$. Let $c := \pi_{j_1} + x_{a_1} - \pi_{i_2}$, and set

$$\pi'_v := \begin{cases} \pi_v & \text{if } v \in E \setminus V(C_2), \\ (\pi_v + c) \bmod T & \text{if } v \in V(C_2). \end{cases}$$

Then the arc $j_1 i_2$ has new tension $x'_{j_1 i_2} = (\pi'_{i_2} - \pi'_{j_1}) \bmod T = (\pi_{i_2} + c - \pi_{j_1}) \bmod T = x_{a_1}$, i.e., it also has the same overlap because all energy arcs a have the same function $\mathrm{overlap}_a$ mapping tensions to overlaps. Moreover, the arc $j_2 i_1$ has tension $x'_{j_2 i_1} = (\pi'_{i_1} - \pi'_{j_2}) \bmod T = (\pi_{i_1} - \pi_{j_2} - c) \bmod T = (\pi_{i_1} - \pi_{j_1} + \pi_{i_2} - \pi_{j_2} - x_{a_1}) \bmod T = x_{a_2}$, i.e., the overlap is also equal. Therefore, together the overlap on the two new arcs is the same as on the two old arcs. So we have found an optimal solution with less cycles, which constitutes a contradiction. ◀

In the second case, we consider a period time that is so large that no energy cycle can exist. This corresponds to an aperiodic timetabling problem.

▶ **Proposition 18.** *Let I be an instance of PESP-Energy on a one-station network with n lines. Let $u^{\max} := \max\{u_a \mid a \in A_{\mathrm{wait}}\} + \max\{t_{a'}^{\min} + t_{a'}^{\max} \mid a' \in A_{\mathrm{energy}}\}$ such that $T > n \cdot u^{\max}$. Then, any matching M^H inducing a Hamiltonian path of maximum weight w.r.t. w is part of an optimal solution $S = (\pi, M^H)$.*

Proof. We show that for every optimal solution $S^{\mathrm{opt}} = (\pi^{\mathrm{opt}}, M^{\mathrm{opt}})$ the set $M^{\mathrm{opt}} \cup A_{\mathrm{wait}}$ contains a Hamiltonian path. By Corollary 11, M^{opt} can be extended to an optimal perfect matching M^p. Let C be an arbitrary directed cycle in $M^p \cup A_{\mathrm{wait}}$. By Proposition 9 we can assume that π^{opt} induces full overlap on all but at most one arc of C. Let a_0 denote this energy arc such that $t_{a_0}^{\min} = \min\{t_a^{\min} \mid a \in C\}$. It holds:

$$\begin{aligned} & t_{a_0}^{\max} + t_{a_0}^{\min} + \sum_{a \in C \cap A_{\mathrm{wait}}} u_a + \sum_{a' \in C \cap A_{\mathrm{energy}} \setminus \{a_0\}} t_{a'}^{\max} \\ & < \sum_{a \in C \cap A_{\mathrm{wait}}} u_a + \sum_{a' \in C \cap A_{\mathrm{energy}}} t_{a'}^{\max} + t_{a'}^{\min} < n \cdot u^{\max} < T. \end{aligned}$$

Therefore, we have $\sum_{a \in C \cap A_{\mathrm{wait}}} u_a + \sum_{a' \in C \cap A_{\mathrm{energy}} \setminus \{a_0\}} t_{a'}^{\max} < T - (t_{a_0}^{\max} + t_{a_0}^{\min})$. Since there is full overlap on all $a' \in C \cap A_{\mathrm{energy}} \setminus \{a_0\}$, we know that the periodic tensions induced by π^{opt} satisfy $x_{a'}^{\mathrm{opt}} \leq t_{a'}^{\max}$ for all these arcs. Hence, for the tension $x_{a_0}^{\mathrm{opt}}$ on a_0 we have $x_{a_0}^{\mathrm{opt}} > t_{a_0}^{\max} + t_{a_0}^{\min}$ as by the cycle periodicity constraints all periodic tensions in C need to sum up to a multiple of T. Therefore, there is no overlap on a_0. By the same argument, every other cycle in $M^p \cup A_{\mathrm{wait}}$ has an arc without overlap. We can remove all these arcs from M^p without reducing the objective value. However, by Theorem 10, any optimal matching

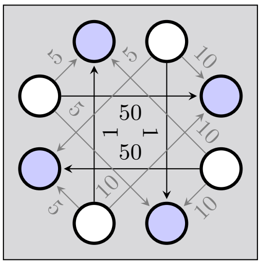


Figure 4 Number of passengers on transfer and waiting activities in instance I_1 from Section 5.

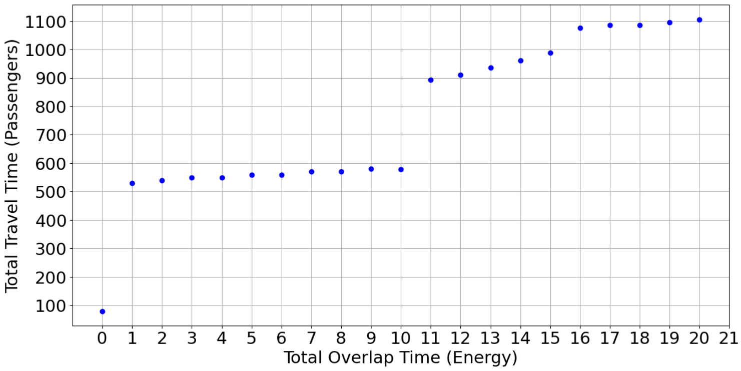


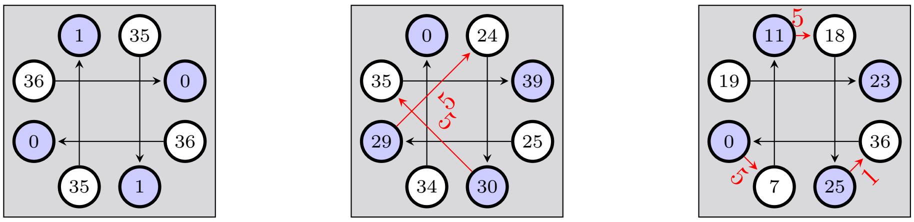
Figure 5 Pareto frontier of instance I_1.

has at least $n-1$ arcs. Therefore, C must be the only cycle in $M^p \cup A_{\text{wait}}$, i.e., it is a Hamiltonian cycle. Moreover, the objective value of S^{opt} is equal to the total weight of $(C \setminus \{a_0\}) \cap A_{\text{energy}}$. An arbitrary maximum-weight Hamiltonian path contains all waiting arcs and then maximizes the weight of the chosen energy arcs. Therefore, it yields the optimal objective value. ◀

5 Examples of Bicriteria Timetables – Numerical Results

In this section, we present some computational results of bicriteria timetabling problems at a single transfer station. We use the MIP formulation from Section 2.3 and solve it with the CPLEX solver on a 13th Gen Intel(R) Core(TM) i5-1335U with 1.30 GHz, 16,0 GB RAM, and a 64-bit processor. We use an ε-constraint method bounding the total brake-traction overlap from below in order to obtain a set of Pareto-optimal solutions. The objective then seeks for the minimal weighted sum of travel times. In the one-station network this equals the weighted sum of the periodic tensions on the waiting and the transfer activities.

The instance I_1 under consideration is based on a one-station network with 2 lines into both directions and a period time of $T = 40$. The acceleration and braking times are all set to $t^{\text{ac}} = 5$ and $t^{\text{br}} = 7$. On the waiting activities, we have the bounds $l = 4$, $u = 8$, and the transfers have a lower bound of $l = 5$ and are non-restricted with $u = 44$. There are no transfers into opposite directions of one line. We assume a symmetric passenger distribution on the arcs, see Figure 4. Figure 5 shows the optimal weighted sum of the periodic tensions (total travel time of the passengers) at this station depending on the required overlap time for the braking and acceleration phases. We observe that the travel times increase with



(a) Overlap ≥ 0
Travel Time: 80.

(b) Overlap ≥ 10
Travel Time: 579.

(c) Overlap ≥ 11
Travel Time: 894.

Figure 6 Exemplary solution structures of I_1. The numbers in the nodes represent the scheduled times of the events. Black arcs represent waiting activities and red arcs represent energy arcs with the numbers indicating the overlap times.

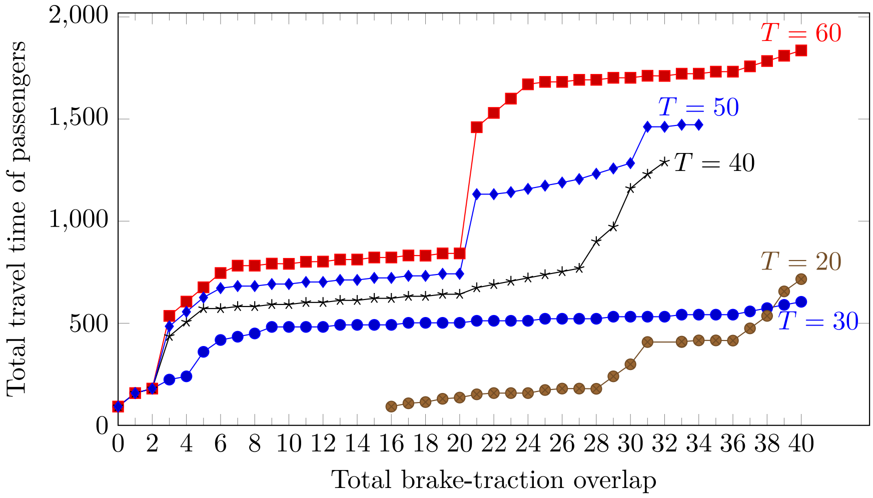


Figure 7 Objective values for instance I_1 with different period times.

increasing required overlap time. Further, there is one huge gap, where the increase from a required overlap time from 10 to 11 results in a huge increase of the total travel time from 579 to 894. In Figure 6 we can see the corresponding timetables for the scenario of no enforced overlap and for a required overlap time of at least 10 and 11. Without an enforced overlap, the timetable has almost a Basel solution structure separately for both the horizontal and the vertical line. If the required overlap is 10, still two trains arrive and depart at almost the same times. This structure disappears for an overlap of at least 11.

Figure 7 shows the Pareto frontiers of instances with the same parameters as I_1 but with varying period time T. We can observe that in general the total travel time of the passengers increases with an increasing period time, which is reasonable as there are transfers with the same number of passengers into both directions of each pair of lines. Further, we can observe that it depends on the period time whether it is possible to attain a maximum overlap of 40 time units. While this is possible for the cases of $T \in \{20, 30, 60\}$, for $T = 50$ we obtain at most 34 time units overlap and for $T = 40$ we obtain a maximum overlap of 32 time units. Due to the cycle periodicity constraints which depend on the period length it is not always possible to obtain full overlap on all chosen energy arcs.

6 Outlook

We have introduced the new periodic timetabling problem PESP-Energy and its bicriteria version. Apart from giving a MIP formulation we characterize the structure of optimal solutions for both single objective problems on a one-station network. On this small network, a polynomial-time algorithm with an additive performance guarantee is obtained for the problem with energy objective. Further, some bicriteria instances on a one-station network were solved numerically and analysed. We plan to continue our work investigating the complexity of the single objective PESP-Energy on a one-station network and developing algorithms for the bicriteria problem on larger networks.

References

1 Ralf Borndörfer, Heide Hoppmann, Marika Karbstein, and Niels Lindner. Separation of cycle inequalities in periodic timetabling. *Discrete Optimization*, 35:100552, February 2020. doi:10.1016/j.disopt.2019.100552.

2 Shuvomoy Das Gupta, J. Kevin Tobin, and Lacra Pavel. A two-step linear programming model for energy-efficient timetables in metro railway networks. *Transportation Research Part B: Methodological*, 93:57–74, November 2016. doi:10.1016/j.trb.2016.07.003.

3 Keivan Ghoseiri, Ferenc Szidarovszky, and Mohammad Jawad Asgharpour. A multi-objective train scheduling model and solution. *Transportation Research Part B: Methodological*, 38(10):927–952, December 2004. doi:10.1016/j.trb.2004.02.004.

4 Paul C. Gilmore and Ralph E. Gomory. Sequencing a one state-variable machine: A solvable case of the traveling salesman problem. *Operations Research*, 12(5):655–679, October 1964. doi:10.1287/opre.12.5.655.

5 Phil G. Howlett, Ian P. Milroy, and Peter J. Pudney. Energy-efficient train control. *Control Engineering Practice*, 2(2):193–200, April 1994. doi:10.1016/0967-0661(94)90198-8.

6 Jinlin Liao, Feng Zhang, Shiwen Zhang, Guang Yang, and Cheng Gong. Energy-saving optimization strategy of multi-train metro timetable based on dual decision variables: A case study of Shanghai metro line one. *Journal of Rail Transport Planning & Management*, 17:100234, March 2021. doi:10.1016/j.jrtpm.2021.100234.

7 Christian Liebchen. *Periodic Timetable Optimization in Public Transport*. PhD thesis, Technische Universität Berlin, 2006.

8 Pengli Mo, Lixing Yang, Andrea D'Ariano, Jiateng Yin, Yu Yao, and Ziyou Gao. Energy-efficient train scheduling and rolling stock circulation planning in a metro line: A linear programming approach. *IEEE Transactions on Intelligent Transportation Systems*, 21(9):3621–3633, September 2020. doi:10.1109/tits.2019.2930085.

9 Michiel A. Odijk. *Construction of Periodic Timetables. Pt. 1. A Cutting Plane Algorithm*, volume 94(61) of *Reports of the Faculty of Technical Mathematics and Informatics*. TU Delft, 1994.

10 Andrés Ramos, María Teresa Pena, Antonio Fernández, and Paloma Cucala. Mathematical programming approach to underground timetabling problem for maximizing time synchronization. *Dirección y Organización*, 35:88–95, June 2008. doi:10.37610/dyo.v0i35.60.

11 Gerben M. Scheepmaker, Rob M. P. Goverde, and Leo G. Kroon. Review of energy-efficient train control and timetabling. *European Journal of Operational Research*, 257(2):355–376, March 2017. doi:10.1016/j.ejor.2016.09.044.

12 Paolo Serafini and Walter Ukovich. A mathematical model for periodic scheduling problems. *SIAM Journal on Discrete Mathematics*, 2(4):550–581, November 1989. doi:10.1137/0402049.

13 Pengfei Sun, Chuanxin Zhang, Bo Jin, Qingyuan Wang, and Haoran Geng. Timetable optimization for maximization of regenerative braking energy utilization in traction network of urban rail transit. *Computers & Industrial Engineering*, 183:109448, September 2023. doi:10.1016/j.cie.2023.109448.

14 Pengling Wang, Yongqiu Zhu, and Francesco Corman. Passenger-centric periodic timetable adjustment problem for the utilization of regenerative energy. *Computers & Industrial Engineering*, 172:108578, October 2022. doi:10.1016/j.cie.2022.108578.

15 Songpo Yang, Feixiong Liao, Jianjun Wu, Harry J.P. Timmermans, Huijun Sun, and Ziyou Gao. A bi-objective timetable optimization model incorporating energy allocation and passenger assignment in an energy-regenerative metro system. *Transportation Research Part B: Methodological*, 133:85–113, March 2020. doi:10.1016/j.trb.2020.01.001.

16 Xin Yang, Bin Ning, Xiang Li, and Tao Tang. A two-objective timetable optimization model in subway systems. *IEEE Transactions on Intelligent Transportation Systems*, 15(5):1913–1921, October 2014. doi:10.1109/tits.2014.2303146.

17 Jiateng Yin, Lixing Yang, Tao Tang, Ziyou Gao, and Bin Ran. Dynamic passenger demand oriented metro train scheduling with energy-efficiency and waiting time minimization: Mixed-integer linear programming approaches. *Transportation Research Part B: Methodological*, 97:182–213, March 2017. doi:10.1016/j.trb.2017.01.001.

A Comparison with the Model of Wang et al.

In the timetable adjustment problem considered by Wang et al. [14], there is an explicitly given set $A_{\text{sab}} \subseteq E_{\text{dep}} \times E_{\text{arr}}$ of activities (j, i) specifying that the acceleration after the departure j (taking time t_j) and the braking before i (taking time t_i) should be synchronized. Their model reads

$$\max O = \sum_{ji \in A_{\text{sab}}} L_{ji} \tag{17}$$

$$\text{s.t. } L^*_{ji} = \min\{\pi_i - \pi_j + \beta_{ji}T,\ \ t_j + t_i - \pi_i + \pi_j - \beta_{ji}T,\ \ t_j,\ \ t_i\} \qquad \forall ji \in A_{\text{sab}} \tag{18}$$

$$M \cdot (\alpha_{ji} + \beta_{ji} - 1) \leq L_{ji} - L^*_{ji} \leq -M \cdot (\alpha_{ji} + \beta_{ji} - 1) \qquad \forall ji \in A_{\text{sab}} \tag{19}$$

$$-M \cdot (\alpha_{ji} + \beta_{ji}) \leq L_{ji} \leq M \cdot (\alpha_{ji} + \beta_{ji}) \qquad \forall ji \in A_{\text{sab}} \tag{20}$$

$$\alpha_{ji} \geq \min\left\{\frac{\pi_i - \pi_j}{M}, \frac{\pi_j - \pi_i + t_j + t_i}{M}\right\} \qquad \forall ji \in A_{\text{sab}} \tag{21}$$

$$\alpha_{ji} \leq 1 + \min\left\{\frac{\pi_i - \pi_j}{M}, \frac{\pi_j - \pi_i + t_j + t_i}{M}\right\} \qquad \forall ji \in A_{\text{sab}} \tag{22}$$

$$\beta_{ji} \geq \min\left\{\frac{\pi_i + T - \pi_j}{M}, \frac{\pi_j - \pi_i - T + t_j + t_i}{M}\right\} \qquad \forall ji \in A_{\text{sab}} \tag{23}$$

$$\beta_{ji} \leq 1 + \min\left\{\frac{\pi_i + T - \pi_j}{M}, \frac{\pi_j - \pi_i - T + t_j + t_i}{M}\right\} \qquad \forall ji \in A_{\text{sab}} \tag{24}$$

$$\alpha_{ji}, \beta_{ji} \in \{0, 1\} \qquad \forall ji \in A_{\text{sab}} \tag{25}$$

$$L_{ji}, L^*_{ji} \in \mathbb{Z} \qquad \forall ji \in A_{\text{sab}} \tag{26}$$

The brake-traction overlap between the events j and i is represented by the variable L_{ji}. The auxiliary variable L^*_{ji} indicates the value of the minimum in the expression for the overlap in Lemma 2, which equals the overlapping time if the overlap is non-empty. This is enforced by Constraint (18). The binary variables α_{ji} and β_{ji} model whether there is some overlap or not: the constraints (21)–(25) ensure that

$$\alpha_{ji} = \begin{cases} 1 & \text{if } \pi_i - \pi_j > 0 \text{ and } \pi_j + t_j > \pi_i - t_i, \\ 0 & \text{if } \pi_i - \pi_j < 0 \text{ or } \pi_j + t_j < \pi_i - t_i, \\ 0 \text{ or } 1 & \text{else} \end{cases} \tag{27}$$

and

$$\beta_{ji} = \begin{cases} 1 & \text{if } \pi_j + t_j > \pi_i + T - t_i, \\ 0 & \text{if } \pi_j + t_j < \pi_i + T - t_i, \\ 0 \text{ or } 1 & \text{else} \end{cases} \tag{28}$$

for all $ji \in A_{\text{sab}}$. Finally, constraints (19) and (20) ensure that the actual overlap L_{ji} is set to L^*_{ji} in the case that $\alpha_{ji} + \beta_{ji} = 1$ and to 0 if $\alpha_{ji} + \beta_{ji} = 0$. Further, the case that $\alpha_{ji} + \beta_{ji} = 2$ is prevented by constraint (19). So it holds

$$\alpha_{ji} + \beta_{ji} \leq 1 \qquad \forall ji \in A_{\text{sab}} \tag{29}$$

Note that the constraint involving a minimum are not linear. To linearize them, we would need to introduce two constraints for each constraint. Further, whenever the minimum bounds from below, we would have to introduce a new binary variable.

Proof of Theorem 3. We begin by showing that the variable L_{ji} in this model measures the same overlap as o_{ji} from PESP-Energy. Hence, we show the following:

1. If there exists an overlap then $L_{ji} = L^*_{ji} = o_{ji}$.
2. If there is no overlap, then $L_{ji} = 0 = o_{ji}$.

The equalities $o_{ji} = L^*_{ji}$ if there is an overlap and $o_{ji} = 0$ else follow from Lemma 2. So we need to investigate the value of L_{ji} for the cases in which there is an overlap (Case 1.1 and 1.2) and in which there is no overlap (Cases 2.1, 2.2 and 2.3).

Case 1.1 [$\pi_j < \pi_i$ and $\pi_j + t_j > \pi_i - t_i$] In this case, $\alpha_{ji} = 1$ by (27) and $\beta_{ji} = 0$ by (28). (23) allows β_{ji} to take the value 0. Hence, $\alpha_{ji} + \beta_{ji} = 1$ and we have $L_{ji} = L^*_{ji}$ by (19).

Case 1.2 [$\pi_j > \pi_i$ and $\pi_j + t_j > \pi_i + T - t_i$] In this case, $\beta_{ji} = 1$ by (28) and $\alpha_{ji} = 0$ by (27). Also by (21) allows α_{ji} to take the value 0. Hence, $\alpha_{ji} + \beta_{ji} = 1$ and we have $L_{ji} = L^*_{ji}$ by (19).

Case 2.1 [$\pi_j < \pi_i$ and $\pi_j + t_j \leq \pi_i - t_i$] In this case, $\alpha_{ji} = 0$ by (27). Further, $\beta_{ji} = 0$ by (28) as $\pi_j + t_j \leq \pi_i - t_i$ implies that $\pi_j + t_j < \pi_i + T - t_i$. Hence, $\alpha_{ji} + \beta_{ji} = 0$ and we have $L_{ji} = 0$ by (19).

Case 2.2 [$\pi_j > \pi_i$ and $\pi_j + t_j \leq \pi_i + T - t_i$] We have that $\alpha_{ji} = 0$ by (27). Further, if $\pi_j + t_j < \pi_i + T - t_i$ then $\beta_{ji} = 0$ by (28). Hence, $\alpha_{ji} + \beta_{ji} = 0$ and we have $L_{ji} = 0$ by (19). On the other hand, if $\pi_j + t_j = \pi_i + T - t_i$ then $\beta_{ji} = 0$ or $\beta_{ji} = 1$ by (29). If $\beta_{ji} = 0$ then it holds, as just discussed, that $L_{ji} = 0$. If $\beta_{ji} = 1$, then it holds that $L_{ji} = L^*_{ji} = \min\{\pi_i - \pi_j + \beta_{ji}T,\ t_j + t_i - \pi_i + \pi_j - \beta_{ji}T,\ t_j,\ t_i\} = \min\{\pi_i - \pi_j + \beta_{ji}T,\ 0,\ t_j,\ t_i\} = 0$ as $t_j, t_i > 0$ and $\pi_i + T - \pi_j > 0$.

Case 2.3 [$\pi_j = \pi_i$] By (27) we get that α_{ji} could be 0 or 1 in this case, also the value for β_{ji} is unclear. Therefore, we rest with the two options $\alpha_{ji} + \beta_{ji} = 0$ and $\alpha_{ji} + \beta_{ji} = 1$. If $\alpha_{ji} + \beta_{ji} = 0$, we know that $L_{ji} = 0$ by constraint (20). If $\alpha_{ji} + \beta_{ji} = 1$, then $L_{ji} = L^*_{ji} = \min\{\pi_i - \pi_j + \beta_{ji}T,\ t_j + t_i - \pi_i + \pi_j - \beta_{ji}T,\ t_j,\ t_i\} = \min\{0,\ t_j + t_i - 0,\ t_j,\ t_i\} = 0$ as $t_j, t_i > 0$ and therefore $t_j + t_i > 0$. $\alpha_{ji} = 1$ by constraint (21) $\alpha_{ji} \leq 1$ by constraint (22) $\beta_{ji} = 0$ by constraint (19). ◀

Solving the Electric Bus Scheduling Problem by an Integrated Flow and Set Partitioning Approach

Ralf Borndörfer
Zuse Institute Berlin, Germany

Andreas Löbel
IVU Traffic Technologies AG, Berlin, Germany

Fabian Löbel[1]
Zuse Institute Berlin, Germany

Steffen Weider
IVU Traffic Technologies AG, Berlin, Germany

Abstract

Attractive and cost-efficient public transport requires solving computationally difficult optimization problems from network design to crew rostering. While great progress has been made in many areas, new requirements to handle increasingly complex constraints are constantly coming up. One such challenge is a new type of resource constraints that are used to deal with the state-of-charge of battery-electric vehicles, which have limited driving ranges and need to be recharged in-service.

Resource constrained vehicle scheduling problems can classically be modelled in terms of either a resource constrained (multi-commodity) flow problem or in terms of a path-based set partition problem. We demonstrate how a novel integrated version of both formulations can be leveraged to solve resource constrained vehicle scheduling with replenishment in general and the electric bus scheduling problem in particular by Lagrangian relaxation and the proximal bundle method.

2012 ACM Subject Classification Mathematics of computing → Mathematical optimization; Mathematics of computing → Network flows; Mathematics of computing → Integer programming; Mathematics of computing → Linear programming; Applied computing → Transportation

Keywords and phrases Electric Bus Scheduling, Electric Vehicle Scheduling, Non-linear Charging, Multi-commodity Flow, Set Partition, Lagrangian Relaxation, Proximal Bundle Method

Digital Object Identifier 10.4230/OASIcs.ATMOS.2024.11

Funding This work has been conducted within the Research Campus MODAL funded by the German Federal Ministry of Education and Research (BMBF) (fund number 05M20ZBM).

1 Introduction

Public transport operators are ramping up the electrification of their bus fleets. Operators in major German cities like Berlin, Hamburg or Munich have pledged to fully electrify their public transport systems by 2030 [25]. Moreover, starting in 2026, it is a legal requirement that 65% of new acquisitions have to have a clean drive train [30]. The European electric bus market is dominated by battery-powered vehicles, depot chargers and fast opportunity chargers at selected terminals [31, 5, 9, 32].

Deploying electric buses has to be planned around their complex energy-cycle. Unlike their diesel counterparts, which can usually drive for an entire day and be fully refueled within minutes upon returning to the depot, electric buses have limited driving ranges and significant recharging times. These limitations are usually exacerbated during summer and winter [31], for instance, enabling air conditioning may reduce a driving range of 250 km down

[1] Corresponding author.

24th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2024).
Editors: Paul C. Bouman and Spyros C. Kontogiannis; Article No. 11; pp. 11:1–11:16
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

to 175 km [35]. As such, electric buses either require short schedules or pre-planned detours and downtime throughout the day to recharge. Moreover, the physics behind recharging batteries lead to non-linear energy models [27], which must be considered for electric vehicle scheduling problems to ensure solutions are actually energy-feasible [24, 26, 22].

This paper is structured as follows: In Section 2 we define the electric bus scheduling problem and in Section 3 we briefly review solution approaches (with non-linear charging) from the literature. In Section 4 we provide a generalized formulation for resource constrained vehicle scheduling by integrating the common flow and path-based models presented in Section 2. In Section 5 we outline how to solve this new formulation leveraging Lagrangian relaxation and the so-called proximal bundle method. For the sake of brevity, we will often refer to older publications for details. Since this algorithm has been in commercial use at a number of public transport operators with partially electrified fleets for a few years now, we can not provide an open source implementation. Finally, we conclude with some computational results in Section 6.

2 Problem Description

Attractive and efficient public transport is contingent on high quality solutions to a number of strategic and operational planning steps, from infrastructure planning, line planning, and timetabling to vehicle scheduling, duty scheduling, crew rostering, and finally real-time disposition [34]. Due to the high computational complexity of each planning task, they are often solved sequentially, even though there are some feedback relationships.

For this paper, we consider the vehicle scheduling planning step for bus systems with (partially) battery-electric fleets, although we believe our results generalize to any electric vehicle scheduling or routing problem. We assume that the infrastructure, bus lines and the timetable have already been fixed, which yield a set of *timetabled passenger trips* $\mathcal{T}$. A *trip* $\tau \in \mathcal{T}$ is the activity of servicing a single repetition of a line from its first to its last stop or terminal. For example, if a bus line has a periodicity of five minutes, then it admits twelve trips per hour. The back direction of a line for this purpose is considered as a separate line.

Each trip needs to be serviced by a bus and any individual bus can not service any trips that happen simultaneously. Moreover, if a bus is scheduled to service two trips in order, it needs to be able to get from the end terminal of the first trip to the start terminal of the second in time to comply with the given fixed timetable. Generally, the set of trips $\mathcal{T}$ together with a relation $\prec$ giving feasible connections called *turns* or *deadheads* can be thought of as a partially ordered set.

The classic (non-electric) *bus scheduling problem* (*BSP*) in its simplest form is to find a cost optimal partition of the trips into chains (subsets of ordered trips), which we call a *vehicle schedule*. An individual chain of trips, i.e., a sequence of trips that can be serviced by the same bus, is a *vehicle course*. The objective function would generally minimize the total required fleet size, i.e., the number of vehicle courses, but also the overall operational expenses.

We are further given a set of depots $\mathcal{D}$ and vehicle types $\mathcal{V}$. Each course has to be assigned a depot the bus has to begin and end service at, and a vehicle type which determines operational costs, but also which trips, deadheads, depots or potentially other infrastructure are accessible. Articulated buses or double-deckers may not fit through every road but some trips have to be serviced by vehicles with a higher capacity for passengers. Moreover, in the electric setting (Section 2.1), we may have vehicle types with different driving ranges, but larger batteries incur higher deployment costs.

We can model *BSP* as a multi-commodity flow problem on a directed acyclic graph where pairs of vehicle type and depot $K = \mathcal{V} \times \mathcal{D}$ serve as the commodities, which we call *plan types*. Every trip admits a node and we have an arc of the form $(\tau_i, \tau_j) \in A$ if and only if $\tau_i \prec \tau_j$, that is, τ_j can be serviced after trip τ_i by the same bus. For every depot $d \in \mathcal{D}$ we add a source node d^{out} and a sink node d^{in} and connect them by pull-out and pull-in arcs to every trip. $K(a)$ then denotes the plan types admissible on arc a, which can be used to control access of particular types to trips. By restricting the pull-in and pull-out arcs at a depot to only the compatible plan types, we further enforce that buses actually return to the depot that they started their course at.

A vehicle course for type $v \in \mathcal{V}$ is then a path between the source and sink nodes of a depot d whose arcs all permit the plan type (v, d). The cost of a vehicle course is simply the sum over arc costs c_a^k of its plan type, which reflect operational expenses. A (binary) multi-commodity flow of minimum cost on this graph then yields a minimum cost vehicle schedule.

Moreover, we have so-called *vehicle-mix constraints* which impose lower and upper bounds on how many courses may be assigned to particular plan types. This is because depots can usually only accommodate a certain number of buses of any particular type, or operators may insist that the fleet composition stay within some parameter. They are given by a family of plan type subsets $\bar{K} \subset 2^K$ and for each $\mathcal{K} \in \bar{K}$, we have bounds $\ell_{\mathcal{K}}$ and $u_{\mathcal{K}}$, as well as coefficients $\kappa_{\mathcal{K}}^k$ per $k \in \mathcal{K}$ such that the weighted sum over all courses of those plan types has to be within $[\ell_{\mathcal{K}}, u_{\mathcal{K}}]$.

In this *BSP* model, trips are generally connected to all reachable trips which happen later within the planning horizon, so a large fraction of the deadheads are *long* in the sense that operators prefer a bus assigned to such a deadhead makes a stopover at a parking facility or depot, where no driver has to be paid to watch over the idle vehicle. In the worst-case, the total number of deadheads is $|\mathcal{T}|\,(|\mathcal{T}|+1)/2$, therefore, long arcs are either dynamically generated on demand while solving *BSP* [20], or they are modeled implicitly via *timelines* (cf. [14, 12]): The planning horizon is discretized and for every time step and parking spot, a node is added to the graph. The nodes of a spot are connected by *idling arcs* in order and there are pull-out and pull-in deadheads between the trips and appropriate timeline nodes. Every long deadhead is then pruned as it now corresponds to a path between a pull-in and a pull-out along a timeline, which decreases the total number of arcs on instances of relevant size [12]. For an example *BSP* graph with five trips, two depots and one parking spot see Figure 1.

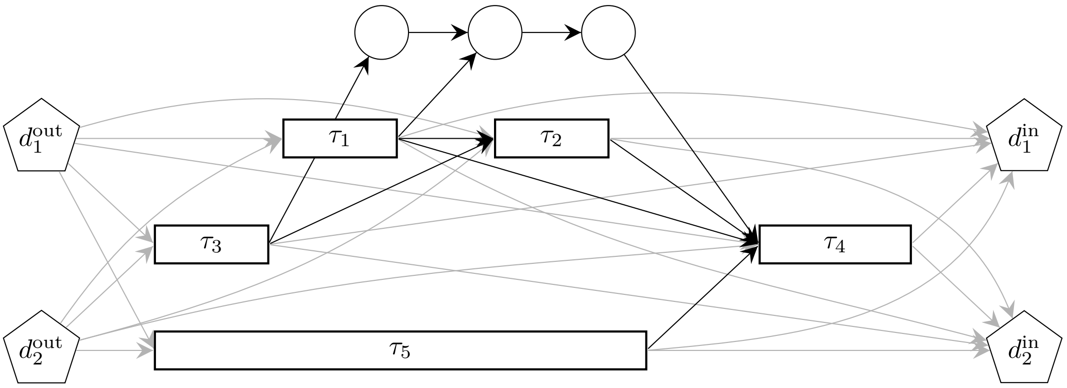


Figure 1 Example *BSP* Graph with five trips, two depots and one parking spot timeline.

The grey arcs are the depot pull-in and pull-out deadheads which only permit plan types of the associated depot. The trip nodes are elongated to indicate their relative start and end times. There is no deadhead from τ_3 to τ_1 as there is no way to make the connection in time. The long deadhead between τ_3 and τ_4 has been pruned in favor of the parking spot timeline on top of the graph, which of course needs larger graphs to be of any advantage. If some trips or deadheads may only be traversed by particular bus types, they must permit only the corresponding plan types, so that the respective nodes and arcs are no longer part of the network of those commodities.

Formulating this multi-commodity flow model of the non-electric *BSP* as an integer linear program, we obtain

$$
\begin{aligned}
(BSP)\quad \min\quad & \sum_{a\in A}\sum_{k\in K(a)} c_a^k x_a^k && && (1)\\
\text{s.t.}\quad & \sum_{a\in\delta^{\text{in}}(n)\colon K(a)\ni k} x_a^k - \sum_{a\in\delta^{\text{out}}(n)\colon K(a)\ni k} x_a^k = 0 && \forall n\in N\setminus\mathcal{D},\ k\in K && (2)\\
& \sum_{a\in\delta^{\text{out}}(\tau)}\sum_{k\in K(a)} x_a^k = 1 && \forall \tau\in\mathcal{T} && (3)\\
& \sum_{a\in\delta^{\text{out}}(n)}\sum_{k\in K(a)} x_a^k \le 1 && \forall n\in N\setminus(\mathcal{D}\cup\mathcal{T}) && (4)\\
& \ell_{\mathcal{K}} \le \sum_{(v,d)\in\mathcal{K}} \kappa_{\mathcal{K}}^{(v,d)} \sum_{a\in\delta^{\text{out}}(d^{\text{out}})} x_a^{(v,d)} \le u_{\mathcal{K}} && \forall \mathcal{K}\in\bar{K} && (5)\\
& x_a^k \in \{0,1\} && \forall a\in A,\ k\in K(a) && (6)
\end{aligned}
$$

where N denotes the entire set of nodes, i.e., it contains $\mathcal{T}$, the depot source and sink nodes and all timeline nodes. Furthermore, $\delta^{\text{in}}(n)$ denotes the set of incoming and $\delta^{\text{out}}(n)$ the set of outgoing deadheads at node n. The binary variables x_a^k indicate whether arc a is selected or *active* for plan type k, i.e., whether a bus of the corresponding type and housed at the corresponding depot traverses it. (2) are flow conservation constraints per commodity, which propagate the selected vehicle type and depot along the flow belonging to a vehicle course. (3) enforces that every trip is covered exactly once and (4) ensures that a parking spot can be used by at most one bus at the same time. (5) are the vehicle-mix constraints.

Note that if $|K| \ge 2$, *BSP* is NP-hard even without any vehicle-mix constraints [1]. Further note that *BSP* is a special case of the vehicle scheduling or routing problem, where vehicles have to visit a set of customers within pre-defined time windows to perform some task of a given duration. The trips correspond to customers with fixed and tight time windows.

Let $M^{\text{F}}x = b$ denote (2) - (5) of the flow formulation in an appropriate matrix notation.

2.1 The Bus Scheduling Problem with Electric Vehicles

The *electric bus scheduling problem* (*EBSP*) extends the *BSP* such that some or all of the bus types are powered by an electric battery. We collect the corresponding plan types in K^E and normalize all battery capacities and energy consumption to a relative driving range in $[0,1]$. Every deadhead arc admits an energy consumption e_a^k per electric plan type, including the consumption of its target trip. We track the remaining driving range via variables y_a at the beginning of every arc, just after the source trip.

Charging takes place at a limited number of charger slots $\mathcal{S}$, so we introduce timelines to track when they are occupied by a bus. We denote those *recharge nodes* s_i by $\bar{\mathcal{S}}$ and add them to N. The corresponding timeline arcs $a(s,i) = (s_{i-1}, s_i)$ are called *recharge arcs* and we denote the set of all recharge arcs by A^E. Along each we can replenish an amount of driving range given by a function $\Delta\zeta_s^k(y_{a(s,i)}, \theta)$ where $y_{a(s,i)}$ is the charge state at the beginning of the recharge arc $a(s,i)$ and θ is the step size of the time discretization. The *charge increment function* $\Delta\zeta_s^k$ depends on the technology employed at charger slot s and the vehicle type of the plan type k. Note that if we are given a charge curve ζ as most of the literature on *EBSP* assumes, that is, a function mapping time spent charging an initially empty battery to the resulting state-of-charge, then it relates to the increment function via $\Delta\zeta(y, \theta) = \zeta(\zeta^{-1}(y) + \theta) - y$. For a homogeneous time step size θ we can just write $\Delta\zeta(y)$.

This yields the, in general non-linear, mixed-integer program

$$(\mathit{EBSP}) \quad \min \sum_{a\in A,\, k\in K(a)} c_a^k x_a^k \tag{7}$$

$$\text{s.t.} \quad M^{\mathrm{F}} x = b \tag{8}$$

$$\sum_{k\in K^E(a)} x_a^k \geq y_a \qquad \forall a \in A \setminus A_{\mathcal{D}}^{\mathrm{out}} \tag{9}$$

$$\sum_{k\in K^E(a)} x_a^k = y_a \qquad \forall a \in A_{\mathcal{D}}^{\mathrm{out}} \tag{10}$$

$$\sum_{\substack{a\in\delta^{\mathrm{in}}(n)\\ k\in K^E(a)}} e_a^k x_a^k = \sum_{a\in\delta^{\mathrm{in}}(n)} y_a - \sum_{a\in\delta^{\mathrm{out}}(n)} y_a \qquad \substack{\forall n\in N\\ n\notin\mathcal{D}\cup\bar{\mathcal{S}}} \tag{11}$$

$$\sum_{\substack{a\in\delta^{\mathrm{in}}(s_i),\\ k\in K^E(a)}} e_a^k x_a^k - \sum_{k\in K^E(a(s,i))} \varphi_{a(s,i)}^k = \sum_{a\in\delta^{\mathrm{in}}(s_i)} y_a - \sum_{a\in\delta^{\mathrm{out}}(s_i)} y_a \qquad \forall a(s,i) \in A^E \tag{12}$$

$$\varphi_{a(s,i)}^k = \Delta\zeta_s^k(y_{a(s,i)}) x_{a(s,i)}^k \qquad \substack{\forall a(s,i)\in A^E,\\ k\in K^E(a(s,i))} \tag{13}$$

$$x_a^k \in \{0,1\} \qquad \forall a \in A,\ k \in K(a) \tag{14}$$

$$y_a \geq 0 \qquad \forall a \in A \tag{15}$$

where $A_{\mathcal{D}}^{\mathrm{out}}$ denotes the set of pull-out arcs at depot nodes, i.e., those arcs that can open new vehicle courses. We retain the *BSP* constraints in (8), but on the graph including charge slot timelines (an example graph would still look like the one in Figure 1, except the timeline may belong to a charge slot). (9) enforces that only active flow-carrying arcs can also have non-zero charge states while (10) requires buses to start service with a full battery. (11) and (12) propagate charge states along active arcs as an energy flow depending on whether any incoming arc is a recharge arc. (13) gives the amount of restored driving range on active recharge arcs depending on the incoming charge state. (14) and (15) are the variable domains. To make (*EBSP*) a linear program we have to linearize the constraint (13), where we refer to our contributions [21] and [22].

2.2 A Set Partition Formulation

It is well-known that *BSP* can be formulated as a set partition problem by applying Dantzig-Wolfe decomposition to the multi-commodity flow formulation. For $k = (v, d) \in K$, let P_k denote all $d^{\mathrm{out}}, d^{\mathrm{in}}$-paths admissible for vehicle type v on the *BSP* graph. Further, let $P = \cup_{k\in K} P_k$. Then, assuming a suitable cost vector $c \in \mathbb{R}^P$ (usually the sum over the arcs on the path), a formulation equivalent to (*BSP*) is

$$\min \sum_{p\in P} c_p x_p \tag{16}$$

$$
\begin{aligned}
\text{s.t.} \quad & \sum_{p \in P : p \ni \tau} x_p = 1 && \forall \tau \in \mathcal{T} && (17)\\
& \sum_{p \in P :\ p \ni n} x_p \leq 1 && \forall n \in N \setminus (\mathcal{D} \cup \mathcal{T}) && (18)\\
\ell^{\mathcal{K}} \leq & \sum_{k \in \mathcal{K}} \sum_{p \in P_k} x_p \leq u^{\mathcal{K}} && \forall \mathcal{K} \in \bar{K} && (19)\\
& x \in \{0,1\}^P && && (20)
\end{aligned}
$$

In theory, it is straightforward to turn this into a formulation for *EBSP*: Let P be the set of all *energy-feasible* paths. A path on the *BSP* graph is energy-feasible if we can insert recharge events such that the battery is never fully depleted and all trips on the path can still be serviced as scheduled. Note that finding a cost-optimal recharge schedule for a given fixed sequence of trips is an instance of the NP-hard *fixed route vehicle charging problem* [24], while testing whether such a sequence is energy-feasible by just charging for as much as possible is polynomially solvable [4]. Further note that the set partition formulation is straightforward to generalize to any resource constrained vehicle scheduling problem with replenishment, like railway operation with maintenance scheduling.

Due to the large number of variables, which in the worst case is one per path on the vehicle scheduling graph, column generation lends itself as the go-to solving approach. The pricing problem is then a *resource constrained shortest path problem with replenishment*, i.e., we need to find resource-feasible paths on the vehicle scheduling graph with negative reduced costs. These paths have to be fit with a cost-optimal resource restoration schedule, which for *EBSP* generally involves evaluating non-linear $\Delta\zeta$.

3 Solution Approaches in the Literature

Electric vehicle routing and scheduling is an active area of research attracting an immense amount of attention. We therefore restrict this literature review to contributions presenting solving approaches for the electric vehicle scheduling problem that can handle non-linear charging explicitly. For an extensive survey on electric vehicle routing and scheduling we refer to [5] and on electric bus scheduling see [28].

An energy state expansion model is proposed in [33], where for each step of a charge state discretization, every node of the vehicle scheduling graph is duplicated. The deadhead arcs connect nodes of appropriate charge states with each other and if there is a recharge window, such a connection can go from a lower to a higher charge state, so $\Delta\zeta$ can be evaluated explicitly per recharge arc. The column generation pricing problem is then a classic shortest path problem on this energy-state-expanded graph. The column generation itself solves the Lagrangian relaxation of a path-based set cover formulation in combination with a rounding heuristic.

In [16] a fully time-and-energy-expanded network is proposed, from which a MILP is derived, where the frequency at which the passenger lines are serviced and the number of chargers are decision variables. The formulation is verified using a commercial MILP solver.

A time-and-energy-expanded network with timelines to track charger slot occupation like in our model is proposed in [3]. Two graphs are obtained by rounding charge states up or down, from which in turn primal and dual bounds can be derived to fuel column generation. A diving heuristic explores the branch-and-bound tree in a depth-first manner to obtain integer solutions.

[24], [7] and [6] are a series of papers which propose a local search to generate a set of candidate vehicle courses that the set partition formulation is solved over. For every candidate vehicle course, the local search has to solve the fixed route vehicle charging problem to determine whether the course is energy-feasible and its cost, so a labeling algorithm and recharge event insertion heuristics are developed. [24] introduces linear spline interpolations of the charge curve ζ from which dominance rules for the labeling method can be derived. [13] extends [7] to consider settings where operators may wish to charge at publicly accessible third-party infrastructure with uncertain availability. The problem is solved by a benders-based branch-and-cut algorithm using a modified version of the labeling algorithm.

Other extensions of [24] are [17] and [10]. [17] develops a label-setting algorithm for the pricing problem of the column generation for the set partition formulation based on recursive functions derived from the linear spline charge curve approximation. This is embedded within a branch-and-price-and-cut framework. [10] extends the model and algorithm from [24] to also include non-linear discharging.

Other papers relying on linear spline approximations of ζ are [36], which presents an adaptive large neighborhood search, [37], which develops a label-setting algorithm considering battery capacity fade, and [38], which also considers capacity fade and develops a MILP with a number of a priori tightening inequalities and runs a commercial solver on it. We assess the previously unknown numerical implications of approximating the charge curve ζ by a linear spline interpolation in [23] and [22].

Lastly, there are a few exact approaches. [26] uses a greedy construction heuristic with backtracking to insert charging events such that arbitrary $\Delta\zeta$ can be considered. [4] proposes a branch-and-check algorithm for factory in-plant electric tow trains. A vehicle schedule whose courses can be made energy-feasible by charging for as much as possible is accepted as the optimal solution, otherwise subtour elimination constraints are introduced to prohibit energy-infeasible courses.

[15] considers an objective function that minimizes the total distance and time spent charging. As such, every recharge event in an optimal solution will only charge for as much as is strictly needed to drive the subsequent trips until the next recharge event or the final depot. One can then use column generation on the set of trip sequences that are energy-feasible without charging and their costs can be derived a priori from the inverse of arbitrary charge curves. It is unclear how this approach can work if the objective does not explicitly minimize the time spent recharging. Operators may not want to strictly minimize charging times due to robustness considerations and active charge management [22]. Nevertheless, generating energy-feasible trip sequences is also a core idea behind our method, which further takes advantage of the easier pricing problem that arises then. Our method could loosely be seen as a generalization of [15] to resource constrained vehicle scheduling with replenishment, multiple depots and vehicle types, and arbitrary linear objective functions. It is also the (to our knowledge) first application of the proximal bundle method [11] to *EBSP*.

4 An Integrated Flow and Set Partition Formulation

In computational experiments, we have found that the set partition formulation for *EBSP* has two undesirable properties, which have also been reported in [15] and we expect should be observable in related problems: For one, there are a large number of very similar columns of negative reduced costs, which cause the master problem to quickly become intractable. Furthermore, the longer the vehicle courses can be and thus have more insertion points for recharge events, the worse the pricing problem performs. In contrast, rounding heuristics to produce integer solutions struggle with the flow formulation. Solutions to the LP-relaxation often collect fractional paths into a single bus, charge its battery, and then fractionally distribute the replenished energy into the network. Vehicle courses derived from a (fractional) path decomposition of the LP solution then often share a single recharge event and it is unclear how to efficiently break this up.

Note that a recent publication [29] shows that electric shortest path with recharging and the corresponding minimum cost flow problem are polynomially solvable if $|K| = 1$, the charge curve ζ is piecewise linear, and every minimum cost subpath in the network is also of minimum energy consumption. However, we believe piecewise linear charge curves to be an inadequate model choice to describe the recharge process [23, 21, 22]. Furthermore, energy-optimal subpaths may not be cost-optimal and vice versa in our application. Some trip to trip arcs involve barely any driving because the bus simply idles at a terminal waiting to service the back direction of the corresponding line. Such a turn has a low energy consumption unlike deadheads that involve proper location changes. But the majority of the operational costs on an arc come from the salary for the driver, so it is possible to have a turn arc that is more expensive than a deadhead arc but that requires less energy.

As we have already alluded to in Section 3, we can make the set partition formulation significantly more tractable by limiting P to those paths that start or end at a depot, where vehicles can be removed from the network, or a facility where consumed resources can be restored. All interior nodes of these paths shall be trips or parking spot timeline nodes and the total resource consumption has to be permissible. In the context of *EBSP*, this means the path can be serviced on a single battery charge. We call these paths *vehicle blocks*. Vehicle courses are then alternating sequences of blocks and restoration or recharge events.

We can then couple the set partition formulation with the flow formulation to have it arrange blocks into vehicle courses by forcing all deadheads contained in active blocks to also be active within the flow. Flow conservation will naturally force deadheads to become active that connect blocks to each other or depots. This yields an integrated flow and set partition formulation for *EBSP*.

We can further completely eliminate all resource related constraints from the formulation by assuming that every block $p \in P$ is serviced by a vehicle that has not consumed any resource yet. Consequently, we have to ensure that all block connections are long enough such that a vehicle's resource state can be fully restored from any initial state. We can enforce this in the timeline model via constraints where an active pull-in onto the timeline prevents pull-outs within the appropriate time window from carrying flow. We then only have to consider the maximum driving range when generating vehicle blocks. This is not a restriction in settings where resource restoration happens in constant time, like regular maintenance or battery swapping, but it does prevent partially recharging a battery for *EBSP* and block connections have to have recharge time windows of length $\zeta^{-1}(1)$.

While this is clearly a major restriction compared to allowing partial charging, we hope that the impact on solution quality is limited by the following observations: On most instances we have encountered so far, recharge events are either a depot charging event with a longer time window, especially in rural settings, or an opportunity charging event with a shorter time window during the turn after a trip. Bus timetables usually tighten their frequency in the morning and late afternoon to evening since the passenger demand is higher during these times. This causes a peak of timetabled trips (see Figure 2) and we need at least as many buses as the largest number of simultaneously scheduled trips as a consequence of Dilworth's theorem. Part of these buses will be idle after the morning peak and they can usually be fully recharged before the afternoon or evening, so the corresponding courses should be mostly unaffected by the restriction.

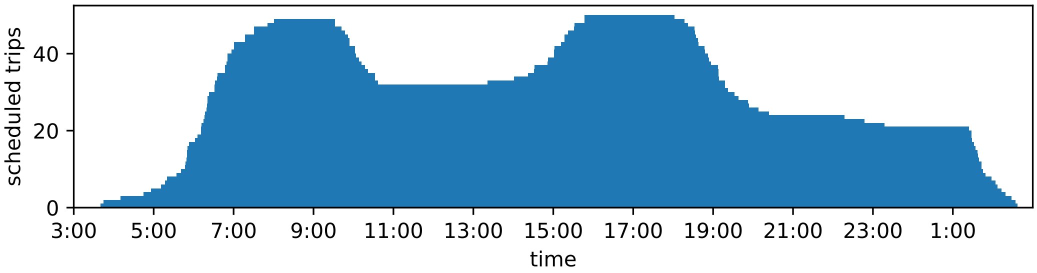


Figure 2 Number of simultaneously scheduled trips of one of our test instances.

Opportunity charging events, as the name suggests, occur during the turns between trips whose terminals are equipped with chargers and a bus can simply recharge there during the mandatory downtime before the next trip. While these events are modeled exactly the same as depot charging via charge slot timelines on the *EBSP* graph, we can efficiently consider them during block generation, unlike depot charging.

Recall that the pricing problem for the full set partition formulation of *EBSP*, where P is the set of energy-feasible vehicle courses, is a resource-constrained shortest path problem with replenishment. This problem is computationally challenging because we can potentially recharge after every trip at any charging facility, for an arbitrary amount of time to an arbitrary state of charge. In particular, the minimum state of charge that the bus has to reach depends on the rest of the path, which in turn depends on how much driving range can actually be restored and the downtime that requires. In fact, as previously mentioned, fitting a cost-optimal recharge schedule to an entirely fixed sequence of trips is NP-hard [24], and the pricing problem to generate vehicle courses is a generalization of this problem.

But if the end terminal of a trip is equipped with an opportunity charger, since the assigned bus is already there, we can simply charge the bus for as long as it can remain depending on whatever trip is put next on the block. Opportunity charging is therefore easy to incorporate into the pricing problem for generating $p \in P$ and we can extend the definition of a vehicle block to allow for opportunity charging whenever a trip terminal is equipped with the necessary infrastructure. This softens our restriction to only apply to depot charging in between blocks, whereas we can consider partial charging for opportunity charging infrastructure as a part of vehicle blocks in P.

We can now give a general *integrated flow and set partition formulation* for *integrated resource constrained vehicle scheduling (with replenishment)* (*IRCVSP*) as

$$
\begin{aligned}
(\mathit{IRCVSP}) \qquad & \min \quad & c^T x & & (21)\\
& \text{s.t.} \quad & M^{\mathrm{F}} x = b & & (22)\\
& & \sum_{p \in P:\, p \ni \tau} w_p = 1 & \quad \forall \tau \in \mathcal{T} & (23)\\
& & \sum_{p \in P_k:\, p \ni a} w_p = x_a^k & \quad \forall a \in A,\ k \in K(a) & (24)\\
& & x_a^k \in \{0,1\} & \quad \forall a \in A,\ k \in K(a) & (25)\\
& & w \in \{0,1\}^P & & (26)
\end{aligned}
$$

where the system (22) is the multi-commodity flow formulation for unconstrained vehicle scheduling as in *BSP*. It enforces the vehicle-mix constraints and that capacities at parking lots and chargers are observed. However, it further needs to guarantee the invariant that vehicles start all blocks with a fully restored resource state by prohibiting block to block connections that are too short. But it is otherwise completely devoid of resource constraints.

Constraints (23) ensure that every trip is covered by exactly one vehicle block and (24) couple the selection of active blocks to that of the active deadheads. Note that if a block contains an opportunity charging event, then that block needs to be coupled to a corresponding charge slot pull-in arc, a number of timeline arcs and a pull-out arc in the flow formulation.

Let $M^{\mathcal{T}}w = \mathbb{1}$ denote (23) and $M^{\text{CF}}x - M^{\text{CP}}w = 0$ denote (24) in matrix notation.

5 Solving *IRCVSP*

Our formulation for *IRCVSP* is superficially similar to standard formulations for the integrated vehicle and duty scheduling problem, where P is defined to be the set of all valid driver duties [34, 2]. A driver duty contains trips and deadheads and has to comply with an underlying vehicle schedule, thus, it can be coupled to a *BSP* flow problem like the vehicle blocks in the *IRCVSP* formulation.

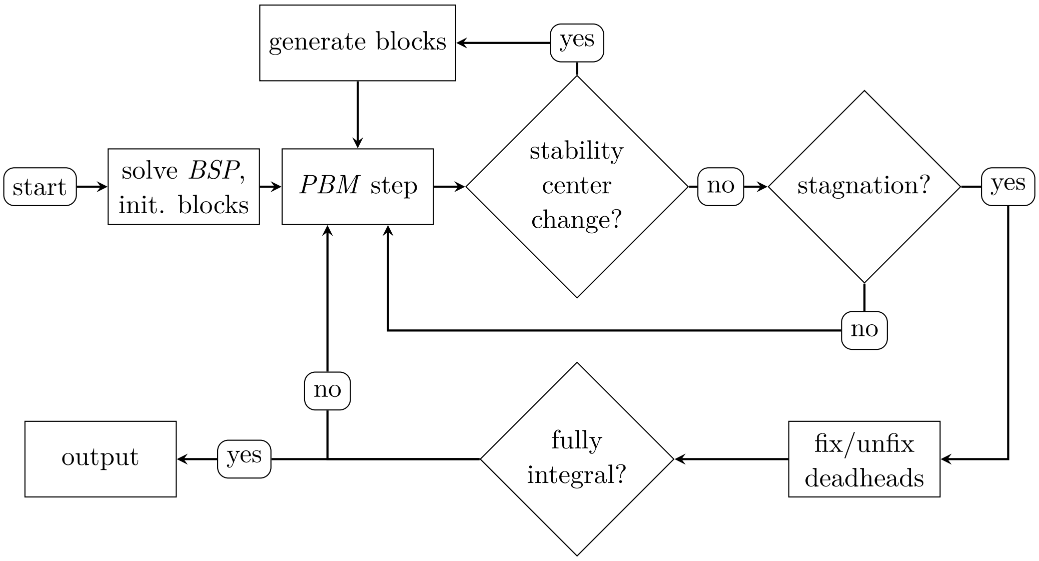


Figure 3 Simplified flow chart of our method to solve *IRCVSP*.

In the case of *EBSP*, there is an intuitive equivalence between drivers and batteries: Both can perform a limited amount of work before they have to be substituted by a fresh replacement. The analogy is literal for battery swapping *EBSP* instances, otherwise replacing a battery by a fresh one means we have to fully recharge it. While driver duties are subject to multiple complicated resource constraints related to working time regulation, we expect similar algorithmic techniques and frameworks used to solve the integrated vehicle and duty scheduling problem to also work for resource constrained vehicle scheduling. In this vein, the algorithm we present here is inspired by our work on the integrated vehicle and duty scheduling problem. Since the algorithmic techniques are quite involved, we only give an outline of the method and describe the necessary modifications to apply it to *EBSP* (see

Figure 3 for a simplified flow chart). For any detailed descriptions omitted here we refer to our previous work [34] and [2]. We further note that the method should generalize to all resource constrained vehicle scheduling problems that fit formulation (*IRCVSP*).

Our method as depicted in Figure 3 has an inner and an outer loop. For the inner loop it relies on the (inexact) *proximal bundle method* (*PBM*) [11] in combination with column generation on the set of vehicle blocks to produce fractional flow values, which are used in a diving heuristic to gradually produce a fully integral flow on the *EBSP* graph by fixing or unfixing arcs as part of the outer loop.

More precisely, we relax the integrality constraints and apply Lagrangian relaxation to the coupling constraints (24) so that the Lagrangian dual

$$\begin{array}{llllllr} \max\limits_{\lambda \in \mathbb{R}^{A \times K(A)}} & \big[\min & \left(c^T - \lambda^T M^{\text{CF}}\right) x & + & \min & \lambda^T M^{\text{CP}}\, w\big] & (27) \\ & \text{s.t.} & M^{\text{F}} x = b & & \text{s.t.} & M^{\mathcal{T}} w = \mathbb{1} & (28) \\ & & x \in [0,1]^{A \times K(A)} & & & w \in [0,1]^P & (29) \end{array}$$

decomposes into the LP-relaxation of the multi-commodity flow and a range-restricted set partition formulation of non-electric *BSP*. The two subproblems are over separate domains and coupled solely via the Lagrange multipliers λ. Therefore, the Lagrangian function (27) is a separable, concave, piecewise linear, and non-smooth function, which can be expressed as $L(\lambda) = f_F(\lambda) + f_P(\lambda)$, where

$$f_F(\lambda) = \min \left\{ \left(c^T - \lambda^T M^{\text{CF}}\right) x \,\middle|\, M^{\text{F}} x = b,\ x \in [0,1]^{A \times K(A)} \right\} \tag{30}$$

and

$$f_P(\lambda) = \min \left\{ \lambda^T M^{\text{CP}} w \,\middle|\, M^{\mathcal{T}} w = \mathbb{1},\ w \in [0,1]^P \right\}. \tag{31}$$

This is exactly the setting for the *PBM* to find the optimal multipliers λ. Given a decomposable concave function such as L, the *PBM* maintains a polyhedral approximation which is iteratively refined along a sequence of so-called stability centers λ_i by evaluating the function components and their subgradients at nearby trial points. Applied to our Lagrangian L from (27), the stability centers λ_i converge towards the optimal multipliers. Furthermore, we can obtain a series that converges towards the optimal primal solution to the LP-relaxation of (*IRCVSP*) from the values x and w that attain f_F and f_P at the trial points [34, 2].

Evaluating f_F and f_P, i.e., solving the LP-relaxations of the flow and the set partition problem for different multipliers, is still computationally challenging and has to be done repeatedly. We therefore solve them approximately, which requires modifications to the *PBM* to still guarantee convergence. For details on this *inexact PBM* for general applications we refer to [11]. How to process approximate evaluations of f_F and f_P is explained in [34, 2].

The flow problem f_F can be (approximately) solved by any appropriate algorithm, we rely on the method described in [18], [19], and [20] as a black-box, which can produce both fractional and integral feasible solutions of high quality as needed. As mentioned before, we have to employ column generation to solve the set partition subproblem f_P, so suppose P^I are the currently selected candidate vehicle blocks from some index set I. If we apply Lagrangian relaxation to this restricted subproblem we obtain

$$\max_{\mu \in \mathbb{R}^{\mathcal{T}}} \left[\mu^T \mathbb{1} + \min_{w_I \in [0,1]^{P^I}} \left(\lambda^T M^{\text{CP}}_{\cdot I} - \mu^T M^{\mathcal{T}}_{\cdot I}\right) w_I \right] \tag{32}$$

where $M_{\cdot I}$ denotes the submatrix made of columns indexed by I. For fixed multipliers λ and μ the minimization is trivial to solve by setting w_i to one if $\lambda^T M^{\text{CP}}_{\cdot\{i\}} \leq \mu^T M^{\mathcal{T}}_{\cdot\{i\}}$ and zero otherwise. For fixed λ from the current *PBM* step of solving (27) we can then use the (exact) *PBM* to determine the optimal μ, which yields an approximation for f_P restricted to P^I and a corresponding argument w_I. It is possible to deduce an approximation of the reduced costs of all vehicle blocks from this by repairing μ^* into a dual feasible, almost optimal solution, see [34] for details. The reduced cost of block i is then $\lambda^T M^{\text{CP}}_{\cdot\{i\}} - (\mu^*)^T M^{\mathcal{T}}_{\cdot\{i\}}$. Since $M^{\mathcal{T}}$ and M^{CP} are simply incidence matrices of which trips and deadheads are contained in which blocks, λ and μ^* yield arc weights which we use for the vehicle block pricing problem as explained in Section 4. It can be solved by standard label-setting techniques. If a deadhead with an opportunity charging window is processed, i.e., a bus can idle at a charger at a trip terminal for a few minutes, we evaluate $\Delta\zeta$ for every respective label. While we eliminate the need to decide when, where and for how long to charge after every trip for the pricing problem and instead offload this decision to the flow subproblem, generating vehicle blocks is still a computationally expensive step, so we only do it when the stability center and thus the candidate trial points of the main *PBM* process changes significantly.

Finally, the lower bounds and (approximated) primal LP-solutions obtained by repeatedly evaluating (27) are used to guide a rounding heuristic to find high quality integer solutions for *IRCVSP*. Once the *PBM* appears to stagnate at the current stability center, we enter the outer loop as indicated in Figure 3 and fix arcs for which x_a is close to 1.0, appropriately propagate this decision through the network and then relaunch the *PBM* algorithm. We dynamically adjust the threshold for when an arc becomes fixed to be more aggressive early on. If a fixing causes the objective to increase by a large margin, the method can backtrack and revert the decision, however, this step is rarely necessary. The final solution is then a feasible binary multi-commodity flow on the vehicle scheduling graph which adheres to all capacity and vehicle-mix constraints, and is straightforward to decompose into individual vehicle courses as explained in Section 2. It is compatible with a selection of energy-feasible blocks which are connected with sufficient downtime to fully recharge the battery, so the vehicle schedule is energy-feasible.

Note that we can obtain an initial solution to start the procedure with by solving the non-electric *BSP* to integrality, then we simply cut the resulting vehicle courses into energy-feasible blocks, which is what the step after “start” in Figure 3 refers to. Throughout the method we also occasionally delete blocks with large reduced cost from P^I to keep the number of candidate blocks tractable.

6 Computational Results

We tested our method on sixteen anonymous real-life *EBSP* instances with sizes depicted in Table 1. Instance G extends F, and I extends H by an additional opportunity fast-charging terminal. Instances K, L and M are variations of the same instance with different charging technology and bus types.

We ran our method until it returned a feasible integral solution and recorded the runtime and objective value. Then, we ran Gurobi 11.0.0 [8] twice on the mixed-integer formulation (7) - (15) for *EBSP* with our charge curve linearization of (13) described in [21] and [22], once with the previously obtained solution and once without any information. After twelve hours of running Gurobi, we recorded the best found objective value, lower bound, and how long it took in the cold-started run to find a solution that was at least as good as the one produced by our method. Note here that our method enforces that the downtime between blocks is large enough to fully recharge an initially empty battery, whereas the MILP permits partial recharge events in the depot. Both permit partial opportunity charging, however. Therefore, the lower bound and best possible objective are in relation to this more flexible charging policy. The results are presented in Table 2 and Figure 4. The experiment was carried out on an AMD EPYC 7542 CPU restricted to two cores and thus four threads per instance.

Table 1 Number of vehicle types, depots, charge slots, timetabled trips and explicitly given deadhead arcs of our test instances. So-called long arcs are given implicitly via pull-in and pull-out arcs at charge slots and parking facilities at the depots.

instance	electric bus types	non-electric bus types	depots	charge slots	trips	deadheads
A	1	0	1	3	121	992
B	1	0	1	2	123	1 078
C	2	0	1	3	146	3 126
D	1	0	1	3	185	1 769
E	1	0	1	8	189	1 977
F	1	0	1	3	232	1 484
G	1	0	1	5	232	2 064
H	1	1	1	6	333	6 363
I	1	1	1	7	333	7 859
J	1	0	1	14	678	15 589
K	1	0	1	43	709	14 431
L	1	0	1	37	709	17 779
M	1	0	1	34	709	21 343
N	2	1	2	12	822	12 390
O	1	1	1	10	837	111 590
P	1	0	1	28	1 207	42 610

Table 2 Runtime comparison between our method and Gurobi on (7) - (15) for *EBSP* without a start solution. The runtime of our method is how long it took to output an integer solution. We compare it to the time Gurobi takes to produce an integer incumbent that is at least as good as the reference solution of our method and the entry of the faster one is highlighted. A value of "-" indicates that Gurobi failed to produce a better solution within twelve hours (43 200 seconds). Gurobi could not produce any feasible integer solution for instances L and P.

runtime (s) of	A	B	C	D	E	F	G	H
our method	**75**	**72**	**413**	165	**196**	371	322	**4 469**
MILP via Gurobi	-	173	4 967	**82**	6 834	**56**	**8**	39 689
	I	J	K	L	M	N	O	P
our method	**6 604**	**1 767**	**1 854**	**5 008**	**2 637**	2 887	29 334	**2 792**
MILP via Gurobi	-	9 929	-	-	-	**297**	**343**	-

Our method can find good solutions faster than Gurobi, as demonstrated on eleven of the sixteen tested instances, with a bias towards the larger ones. On six instances it produces a solution in less than two hours that Gurobi can not beat within twelve. In particular, Gurobi

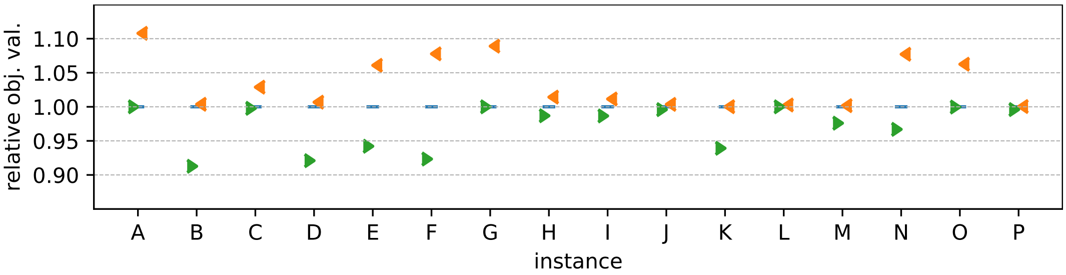


Figure 4 Relative objective values. The best objective value we obtained after letting Gurobi run for twelve hours, both with and without a start solution, is the baseline at 1.0. The best lower bound is given as a relative value by the green triangles pointing to the right. The objective value of the solution produced by our method is given as a relative value by the orange triangles pointing to the left.

fails to produce any feasible vehicle schedule for instances L and P, whereas our method produces solutions that are almost optimal even under the model where partial charging in the depot is allowed, as can be seen in Figure 4.

Integer heuristics used by Gurobi can produce good schedules faster than the overhead of our method permits for the small instances D, F, and G, but those heuristics are not consistent on our test set as can be seen for other small instances A, C, and E. Among the larger instances, N and especially O are the outliers for which our method is significantly slower than Gurobi. Examining the vehicle schedules we see that the problem is the restriction to full recharge windows between blocks. Gurobi can find the optimal solution for O, which admits one hundred and three recharge events, whereas our method produces a solution with sixteen. Most of these recharge events in the optimal solution are short and merely top off an almost fully charged bus, i.e., they are like opportunity recharge events, except the bus takes a small detour of about five minutes to the depot charger. In our data, this depot charger is of course not flagged as opportunity charging infrastructure and is therefore not considered by the vehicle block pricing algorithm. Our method then apparently struggles to produce compatible blocks that allow for full recharge windows in between. We make a similar observation for instance N.

Further examining the objective values and lower bounds in Figure 4, we see that our method solves J, L, and P almost optimally and additionally attains the best found solution for B, K, and M. At worst, it is within 11% of the best found solution and 15% of the best lower bound. Lastly, note that a sequential approach of our method and then Gurobi on the MILP solved A, C, G, J, L, and P to optimality within a total time limit of fourteen hours, whereas Gurobi did not produce any feasible solution for L and P on its own.

References

1 Alan A. Bertossi, Paolo Carraresi, and Giorgio Gallo. On Some Matching Problems Arising in Vehicle Scheduling Models. *Networks*, 17:271–281, 1987.

2 Ralf Borndörfer, Andreas Löbel, and Steffen Weider. A Bundle Method for Integrated Multi-Depot Vehicle and Duty Scheduling in Public Transit. In Mark Hickman, Pitu Mirchandani, and Stefan Voß, editors, *Computer-aided Systems in Public Transport*, volume 600, pages 3–24, 2008.

3 Marelot H. de Vos, Rolf N. van Lieshout, and Twan Dollevoet. Electric Vehicle Scheduling in Public Transit with Capacitated Charging Stations. *Transportation Science*, 2023:1–16, 2023. `doi:10.1287/trsc.2022.0253`.

4 Heiko Diefenbach, Simon Emde, and Christoph H. Glock. Multi-depot electric vehicle scheduling in in-plant production logistics considering non-linear charging models. *European Journal of Operational Research*, 306(2):828–848, 2023. doi:10.1016/j.ejor.2022.06.050.

5 Tomislav Erdelić and Tonči Carić. A Survey on the Electric Vehicle Routing Problem: Variants and Solution Approaches. *Journal of Advanced Transportation*, 2019:1–48, May 2019. doi:10.1155/2019/5075671.

6 Aurélien Froger, Ola Jabali, Jorge E. Mendoza, and Gilbert Laporte. The Electric Vehicle Routing Problem with Capacitated Charging Stations. *Transportation Science*, 56(2):460–482, 2022. doi:10.1287/trsc.2021.1111.

7 Aurélien Froger, Jorge E. Mendoza, Ola Jabali, and Gilbert Laporte. Improved formulations and algorithmic components for the electric vehicle routing problem with nonlinear charging functions. *Computers & Operations Research*, 104:256–294, 2019. doi:10.1016/j.cor.2018.12.013.

8 Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2024. URL: https://www.gurobi.com.

9 Dominic Jefferies and Dietmar Göhlich. A Comprehensive TCO Evaluation Method for Electric Bus Systems Based on Discrete-Event Simulation Including Bus Scheduling and Charging Infrastructure Optimisation. *World Electric Vehicle Journal*, 11, August 2020. doi:10.3390/wevj11030056.

10 Yong Jun Kim and Byung Do Chung. Energy consumption optimization for the electric vehicle routing problem with state-of-charge-dependent discharging rates. *Journal of Cleaner Production*, 385:135703, 2023. doi:10.1016/j.jclepro.2022.135703.

11 Krzysztof C. Kiwiel. A Proximal Bundle Method with Approximate Subgradient Linearizations. *SIAM Journal on Optimization*, 16(4):1007–1023, 2006. doi:10.1137/040603929.

12 Natalia Kliewer, Taïeb Mellouli, and Leena Suhl. A time–space network based exact optimization model for multi-depot bus scheduling. *European Journal of Operational Research*, 175(3):1616–1627, 2006. doi:10.1016/j.ejor.2005.02.030.

13 Nicholas D. Kullman, Justin C. Goodson, and Jorge E. Mendoza. Electric Vehicle Routing with Public Charging Stations. *Transportation Science*, 55(3):637–659, 2021. doi:10.1287/trsc.2020.1018.

14 Achim Lamatsch. An Approach to Vehicle Scheduling with Depot Capacity Constraints. In Martin Desrochers and Jean-Marc Rousseau, editors, *Computer-Aided Transit Scheduling*, pages 181–195, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

15 Chungmok Lee. An exact algorithm for the electric-vehicle routing problem with nonlinear charging time. *Journal of the Operational Research Society*, 72(7):1461–1485, 2021. doi:10.1080/01605682.2020.1730250.

16 Lu Li, Hong K. Lo, and Feng Xiao. Mixed bus fleet scheduling under range and refueling constraints. *Transportation Research Part C: Emerging Technologies*, 104:443–462, 2019. doi:10.1016/j.trc.2019.05.009.

17 Yijing Liang, Said Dabia, and Zhixing Luo. The Electric Vehicle Routing Problem with Nonlinear Charging Functions, 2021. arXiv:2108.01273.

18 Andreas Löbel. *Optimal Vehicle Scheduling in Public Transit.* PhD thesis, Technische Universität Berlin, 1997.

19 Andreas Löbel. Vehicle Scheduling in Public Transit and Lagrangean Pricing. *Management Science*, 44(12-1):1637–1649, 1998.

20 Andreas Löbel. Solving Large-Scale Multi-Depot Vehicle Scheduling Problems. *Computer-Aided Transit Scheduling*, pages 193–220, 1999.

21 Fabian Löbel, Ralf Borndörfer, and Steffen Weider. Non-Linear Charge Functions for Electric Vehicle Scheduling with Dynamic Recharge Rates. In Daniele Frigioni and Philine Schiewe, editors, *23rd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2023)*, volume 115 of *Open Access Series in Informatics (OASIcs)*, pages 15:1–15:6, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/OASIcs.ATMOS.2023.15.

22 Fabian Löbel, Ralf Borndörfer, and Steffen Weider. Electric Bus Scheduling with Non-Linear Charging, Power Grid Bottlenecks and Dynamic Recharge Rates. Technical report, Zuse-Institute Berlin, Takustraße 7, 14195 Berlin, Germany, 2024. (pre-print, submitted for review). doi:10.48550/arXiv.2407.14446.

23 Fabian Löbel, Ralf Borndörfer, and Steffen Weider. Non-linear Battery Behavior in Electric Vehicle Scheduling Problems. In Guido Voigt, Malte Fliedner, Knut Haase, Wolfgang Brüggemann, Kai Hoberg, and Joern Meissner, editors, *Operations Research Proceedings 2023*, Lecture Notes in Operations Research. Springer Cham, 2024.

24 Alejandro Montoya, Christelle Guéret, Jorge E. Mendoza, and Juan G. Villegas. The electric vehicle routing problem with nonlinear charging function. *Transportation Research Part B: Methodological*, 103:87–110, 2017. Green Urban Transportation. doi:10.1016/j.trb.2017.02.004.

25 Michael Neißendorfer. Kiepe Electric lädt E-Busse mit Bahnstrom besonders schnell und effizient. *Nahverkehrs-praxis*, 6-2023:60–61, 2023. (German).

26 Nils Olsen and Natalia Kliewer. Scheduling Electric Buses in Public Transport: Modeling of the Charging Process and Analysis of Assumptions. *Logistics Research*, 13(4), 2020. doi:10.23773/2020_4.

27 Samuel Pelletier, Ola Jabali, Gilbert Laporte, and Marco Veneroni. Battery degradation and behaviour for electric vehicles: Review and numerical analyses of several models. *Transportation Research Part B: Methodological*, 103:158–187, 2017. doi:10.1016/j.trb.2017.01.020.

28 Shyam S.G. Perumal, Richard M. Lusby, and Jesper Larsen. Electric bus planning & scheduling: A review of related problems and methodologies. *European Journal of Operational Research*, 2022. doi:10.1016/j.ejor.2021.10.058.

29 Haripriya Pulyassary, Kostas Kollias, Aaron Schild, David Shmoys, and Manxi Wu. Network Flow Problems with Electric Vehicles. In Jens Vygen and Jarosław Byrka, editors, *Integer Programming and Combinatorial Optimization*, pages 365–378, Cham, 2024. Springer Nature Switzerland. doi:10.1007/978-3-031-59835-7_27.

30 Steffen Schulze and Jascha Lackner. Mehr saubere Busse im ÖPNV. *Nahverkehrs-praxis*, 6-2023:50–52, 2023. (German).

31 Ryan Sclar, Camron Gorguinpour, Sebastian Castellanos, and Xiangyi Li. Barriers to Adopting Electric Buses. Technical report, World Resource Institute, 10 g street ne, suite 800, Washington, DC 20002, USA, 2019. last accessed June 2024. URL: https://www.sustainable-bus.com/wp-content/uploads/2019/05/barriers-to-adopting-electric-buses.pdf.

32 Michael Sievers and Andreas Laumen. Großes Potenzial für den Einsatz von eBussen. *Nahverkehrs-praxis*, 6-2023:42–73, 2023. (German).

33 M. E. van Kooten Niekerk, J. M. van den Akker, and J. A. Hoogeveen. Scheduling electric vehicles. *Public Transport*, 9:155–176, 2017. doi:10.1007/s12469-017-0164-0.

34 Steffen Weider. *Integration of Vehicle and Duty Scheduling in Public Transport*. PhD thesis, Technische Universität Berlin, 2007.

35 Weitiao Wu, Yue Lin, Ronghui Liu, and Wenzhou Jin. The multi-depot electric vehicle scheduling problem with power grid characteristics. *Transportation Research Part B: Methodological*, 155:322–347, 2022. doi:10.1016/j.trb.2021.11.007.

36 Aijia Zhang, Tiezhu Li, Ran Tu, Changyin Dong, Haibo Chen, Jianbing Gao, and Ye Liu. The Effect of Nonlinear Charging Function and Line Change Constraints on Electric Bus Scheduling. *Promet - Traffic & Transportation*, 33(4):527–538, 2021. doi:10.7307/ptt.v33i4.3730.

37 Le Zhang, Shuaian Wang, and Xiaobo Qu. Optimal electric bus fleet scheduling considering battery degradation and non-linear charging profile. *Transportation Research Part E: Logistics and Transportation Review*, 154:102445, 2021. doi:10.1016/j.tre.2021.102445.

38 Yu Zhou, Qiang Meng, and Ghim Ping Ong. Electric Bus Charging Scheduling for a Single Public Transport Route Considering Nonlinear Charging Profile and Battery Degradation Effect. *Transportation Research Part B: Methodological*, 159:49–75, 2022. doi:10.1016/j.trb.2022.03.002.

Towards an Optimization Pipeline for the Design of Train Control Systems with Hybrid Train Detection

Stefan Engels[1]
Chair for Design Automation, Technical University of Munich, Germany

Robert Wille
Chair for Design Automation, Technical University of Munich, Germany
Software Competence Center Hagenberg GmbH (SCCH), Austria

Abstract

Increasing the capacity of our railway infrastructure will become more and more essential in coping with the need for sustainable transportation. This can be achieved by intelligently implementing train control systems on specific railway networks. Methods that automate and optimize parts of this planning process are of great interest. For control systems based on hybrid train detection, such optimization tasks simultaneously involve routing and block layout generation. These tasks are already complex on their own; hence, a joint consideration often becomes infeasible. This work-in-progress paper proposes an idea to tackle the corresponding complexity. To this end, we present a pipeline that allows to *sequentially* handle corresponding optimization tasks in a less complex fashion while generating results that remain (close to) optimal. Results from an initial case study showcase that this approach is, indeed, promising. A prototypical implementation is included in the open-source *Munich Train Control Toolkit* available at `https://github.com/cda-tum/mtct`.

2012 ACM Subject Classification Applied computing → Transportation

Keywords and phrases ETCS, MILP, Design Automation, Hybrid Train Detection

Digital Object Identifier 10.4230/OASIcs.ATMOS.2024.12

Category Short Paper

Supplementary Material *Software (Source Code)*: `https://github.com/cda-tum/mtct`
archived at `swh:1:dir:37ddd0d5e96dd5ff76d63d29d27d9aac64400d37`

1 Introduction

The demand for railroads is increasing as sustainable transportation becomes more and more important. However, the capacity of existing railway infrastructure is limited. In addition to building new tracks, increasing the capacity of existing lines is crucial to satisfy the growing demand, e.g., by utilizing more efficient train control systems. For reasons of compatibility, international train control systems have been specified in a standardized fashion, e.g., through the *European Train Control System* (ETCS), *Chinese Train Control System* (CTCT), and *Positive Train Control* (PTC) [10]. Each of them comes in different variants. More sophisticated levels allow shorter train following times (headways), and by this, increase the capacity while, at the same time, maintaining a high level of safety.

During the planning process, design choices have to be made that might influence the outcome. Much of the planning relies on manual processes and the work experience of the involved personnel, an expensive and error-prone endeavor. Automating and optimizing specific steps to reduce these costs and ensure the best operational outcome is of great interest [3]. Accordingly, methods have been developed that optimize train operation, such

[1] Corresponding author

24th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2024).
Editors: Paul C. Bouman and Spyros C. Kontogiannis; Article No. 12; pp. 12:1–12:6
OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

as creating timetables and routing [2]. Optimal routing has recently been considered for trains operating under so-called *Moving Block* [12, 9, 6]. In addition, classical control systems rely on separating the network into blocks, requiring physical hardware for each of them. Generating optimal block layouts focused on optimizing general performance indicators independent of specific schedules [7, 13].

Alternatively, modern specifications relying on *Hybrid Train Detection* allow the introduction of purely virtual (sub-)sections. At least in theory, they allow the flexible adjustment of the layout, leading to new design objectives to be optimized [4]. To the best of our knowledge, algorithms tailored to hybrid train detection have first been considered in [14, 11]. While these initial solutions neglect significant modeling details, a more accurate solution method has been introduced in [5]. Unfortunately, these solutions do not scale well. A primary reason for that might be because they combine multiple complex objectives in one task.

In this work-in-progress paper, we propose an optimization pipeline that considers the resulting sub-tasks sequentially. This allows for solving these problems in a less complex fashion while still generating (close to) optimal solutions. Results obtained by initial case studies confirm these premises. A prototypical implementation of the proposed idea is available open-source as part of the *Munich Train Control Toolkit* at `https://github.com/cda-tum/mtct`.

The remainder of this work is structured as follows. Sec. 2 summarizes the relevant background, namely principals of train control systems in Sec. 2.1 and resulting design tasks in Sec. 2.2. Afterward, Sec. 3 describes the proposed optimization pipeline and constitutes the main contribution of this work in progress. A short case study in Sec. 4 demonstrates that this approach is promising, and Sec. 5 concludes this paper.

2 Background

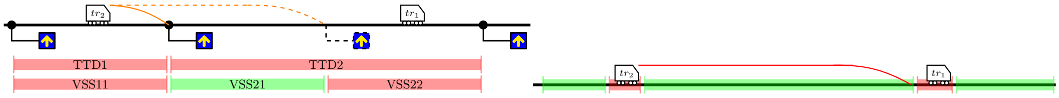
2.1 Moving Block and Hybrid Train Detection

Due to long braking distances, it is not feasible for trains to operate on sight. Instead, signaling systems are implemented to ensure safe operation. Classical control systems divide the network into fixed block sections. A train cannot enter a section that is already occupied by another train. Physical *Trackside Train Detection* (TTD) hardware (e.g., axle counters) at the section borders is used to detect the position of trains.

Modern specifications require trains to report their exact positions with certainty. By doing so, TTD hardware is no longer needed. Ideally, a *Moving Block* control system can be implemented in which trains follow each other at their (absolute) braking distance (similar to car traffic) without the need to define block sections.

However, such a system might impose practical problems, especially on lines with mixed traffic where some trains might not be equipped with train integrity monitoring systems to safely report their positions to the control system [1]. As a “compromise” *Hybrid Train Detection* has been specified. For this, existing TTD sections are separated into smaller *Virtual Subsections* (VSS) without the need for additional hardware. This allows for shorter headway times. At the same time, the original TTD sections serve as a backup.

► **Example 1.** Consider the scenario shown in Fig. 1 with two trains. In Fig. 1a, the network is divided into block sections TTD1 and TTD2. Because TTD2 is occupied by train tr_1, the following train tr_2 can only advance to the end of TTD1 (solid orange line). Using hybrid train detection, TTD2 might be separated into two virtual subsections. Because of this, train tr_2 is authorized to move until the end of VSS21 (dashed orange line). On the contrary, there are no block sections under moving block control. In Fig. 1b, train tr_2 can advance up to the end of train tr_1 minus a small safety buffer.



(a) Hybrid Train Detection. **(b)** Moving Block Control.

Figure 1 Schematic drawings of various ETCS levels.

2.2 Resulting Design Tasks

In the following, we focus on control systems with hybrid train detection. At least in theory, the virtual block layout can be chosen flexibly. This allows, for the first time, the VSS to be adjusted depending on a specific train schedule. Hence, new design tasks result that utilize this additional degree of freedom [4].

In general, the question is how to separate a given layout into VSS sections to obtain the best operational outcome. Some objectives might be determining a minimal number of subsections to make a previously infeasible timetable possible, minimizing runtimes using a predefined number of VSS, or maximizing the throughput of additional (e.g., freight) trains. Nevertheless, the focus is to achieve this operational benefit by intelligently defining a (virtual) block layout. It can be shown that all of those tasks are NP-hard, even if the routing aspect is fixed. For more details (which are out of the scope of this paper), we refer to previous work [4].

3 Towards an Optimization Pipeline

The design tasks reviewed above have in common that they consist of two main parts, namely train routing and placement of VSS sections. At the same time, they affect each other. The feasibility of a routing depends on the chosen VSS layout, and the necessity of subsections depends on the routing. Both tasks are NP-hard already on their own; hence, a joint consideration often makes solving these tasks infeasible. To cope with the corresponding complexity, we propose using an optimization pipeline to solve the two aspects sequentially while still getting (close to) optimal solutions.

To this end, we use the following observation: Moving block control can be seen as classic block signaling where each section is infinitesimally small. In particular, a routing under moving block is likely feasible if a sufficient amount of block sections is defined. Nevertheless, finding such a routing is easier on moving block systems because the optimization model does not have to generate a (virtual) block layout simultaneously. Moreover, there is already promising work for time-optimal routing on moving block controlled networks [12, 9, 6]. Thus, we propose a two-step approach, quasi an "optimization pipeline":

1. The trains are routed as if they were to operate under moving block control using the approaches mentioned above.
2. The routing obtained from Step 1 is fixed, and VSS sections are then generated based on this assumption.

Based on the above reasoning, we conjecture that Step 1 will likely choose the same routes as a combined optimization model would have produced (even though there is no theoretical guarantee). In this case, Step 2 outputs the same optimal solution, but the sequential approach substantially reduces the complexity compared to the joint consideration.

In order to implement that idea, we can utilize the approach proposed in [5], which is based on a *Mixed Integer Linear Program* (MILP) that (in principle) jointly models Steps 1 and 2. At the same time, this approach offers an option to additionally constrain trains

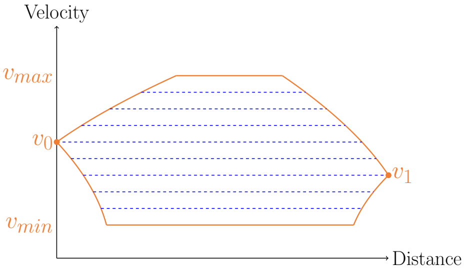


Figure 2 Velocity Profiles.

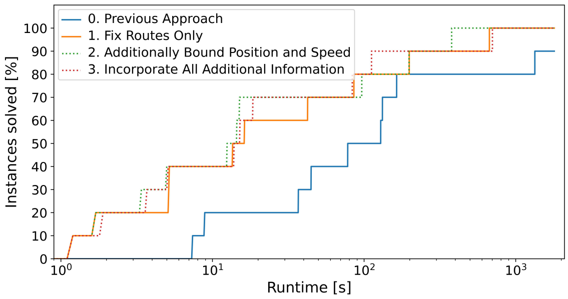


Figure 3 Experimental Evaluation.

to use predefined routes; hence, it can be used in Step 2. In [5], it was already shown that this option is beneficial under the assumption that these routes are available “for free”. Unfortunately, the question of how (and at what additional cost) to obtain this information has not been investigated before.

Given a solution obtained by Step 1, we can extract the used edges and even more information to narrow down the search space and guide the optimization algorithm in Step 2. To this end, observe that the approaches in [12, 9, 6] (which can be used for Step 1) only model the times and velocities when entering and leaving specific track segments. Say a train enters a given track segment with velocity v_0 and exits at speed v_1. The intermediate positions and velocities can only be interpolated and might not be uniquely defined. To this end, consider Fig. 2. The orange lines denote the (two) extreme velocity profiles that might occur on the track segment and correspond to the min- and max-time profiles in [12]. If we assume that trains only accelerate/decelerate close to the ends and travel at constant line speed in between, we obtain the dashed blue profiles. Doing so allows assigning a well-defined approximate velocity profile for any possible timing. Keeping this in mind, we concretize:

- *Fix Train Orders:* To ensure train separation, every formulation has to somehow model in which order trains traverse specific track segments. This usually adds complexity to the underlying model. However, we can extract those train orders from a Step 1 solution and fix it for Step 2, which reduces the feasible region, prunes the search space, and might lead to faster solving times.
- *Fix Train Positions and Velocities:* Solving Step 2 with the method proposed in [5], position and velocity are modeled at a discretized set of time points. Using the above observation, we can extract lower and upper bounds at every time point using the extreme profiles and add this information as constraints. Theoretically, this could cut off the optimal solution. However, we conjecture this to be unlikely due to the aforementioned reasoning that train routes are likely equivalent under both controlling principles. To be on the safe side, we add an additional tolerance of the distance traveled in one time step to reduce a possibly negative effect of discretization errors.
- *Hint Approximate Train Positions:* Using the specific timings from Step 1, we can map precisely one of the approximate velocity profiles mentioned above. At any time, we can easily calculate exact positions and velocities; however, the actual trajectory might differ. Because of this, we are not sure enough to add this information using equality constraints. However, we can pass these as a variable hint to the MILP solver, indicating that we believe the optimal solution is close to that approximated trajectory. Some solvers, e.g., Gurobi, can use this information to speed up the optimization process by adapting heuristics and branching decisions [8].

Overall, the above ideas allow for a sequential (rather than joint) consideration of the corresponding design aspects. This may provide the path towards efficiently handling those design tasks while still maintaining (close to) optimal results.

4 Case Study: Generation of Minimal VSS Layouts

To preliminary evaluate the proposed approach, we tested it on one of the more straightforward design tasks: generating minimal VSS layouts to make a specific timetable possible. Our implementation is based on [12, 6] for Step 1 and on [5] for Step 2, which was extended to include the additional information described in Sec. 3. The code is available as an open-source implementation on GitHub at `https://github.com/cda-tum/mtct`. We used the same benchmark as in [5] and an Intel(R) Xeon(R) W-1370P system using a 3.60GHz CPU (8 cores) and 128GB RAM running Ubuntu 20.04 and Gurobi version 11.0.2 [8].

The resulting runtimes[2] are plotted in Fig. 3 (see previous page). The x-axis shows timeouts in seconds, whereas the y-axis represents the percentage of instances solved within the given time or faster. By design, all lines are monotonously increasing, and being on the left/top is considered to be "better". For comparison, we solved the benchmark using the previous approach [5], which jointly considers all decision aspects. Additionally, all instances were solved using the proposed sequential approach. In Step 2, three variants have been considered, namely,

1. only fixing the routes (i.e., used edges) without any additional information,
2. additionally, constraining position and speed at every time step, and,
3. finally, incorporating all information described in Sec. 3.

The depicted runtimes are total times, i.e., the sum of both Step 1 and Step 2, as well as model creation times.

Overall, these initial case studies clearly show the benefit of the proposed pipeline. Even though this includes two optimization steps, it is consistently and significantly faster than the previous approach. On the other hand, we can observe that most of this improvement is due to the separation of routing and VSS placement (orange solid line). The additional information described in Sec. 3 (dotted lines) seem to further improve the runtime in most cases; however, the difference is not as big. Moreover, it is not immediately apparent which of the described information are best to be included. Still, we can conclude that adding all additional information is never a bad idea. It is just that, in some cases, almost the entire runtime benefit might be due to fixing edges and not due to additional information. Finding the best set of parameters within the proposed optimization pipeline is left to future research.

5 Conclusions

With this work, we proposed a step towards an efficient optimization pipeline for designing railway networks based on train control with hybrid train detection. We demonstrate how routing information from a different control principle, namely, moving block, can significantly simplify the optimization model. Even though the resulting approach consists of two optimization steps, the runtime is significantly reduced. The resulting prototypical

[2] We do not show the objective values in detail. The optimal solution was returned independently of the chosen algorithm in all but one instance. In this one case, fixing position bounds led to an increase in VSS sections from 6 to 13, even though the route itself was optimal. All other parameters did not have any effect on the objective. Overall, this shows that the proposed approach often yields (close-to) optimal results (even though there is no theoretical guarantee).

implementation is available open-source and included within the Munich Train Control Toolkit at `https://github.com/cda-tum/mtct`. Future work focuses on a more sophisticated implementation and evaluation of the idea presented in this paper. This includes the extension to more complex design tasks and objectives as well as the development of algorithms tailored to this framework to use more information on the relevant problem structure already at the core of their development.

References

1 Maarten Bartholomeus, Laura Arenas, Roman Treydel, Francois Hausmann, Nobert Geduhn, and Antoine Bossy. ERTMS Hybrid Level 3. *SIGNAL + DRAHT (110) 1+2*, 2018. URL: `https://www.eurailpress.de/fileadmin/user_upload/SD_1_2-2018_Bartholomaeus_ua.pdf`.

2 Ralf Borndörfer, Torsten Klug, Leonardo Lamorgese, Carlo Mannino, Markus Reuther, and Thomas Schlechte, editors. *Handbook of Optimization in the Railway Industry*. Springer, 2018. `doi:10.1007/978-3-319-72153-8`.

3 Stefan Dillmann and Reiner Hähnle. Automated planning of ETCS tracks. In *Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification*. Springer, 2019. `doi:10.1007/978-3-030-18744-6_5`.

4 Stefan Engels, Tom Peham, Judith Przigoda, Nils Przigoda, and Robert Wille. Design tasks and their complexity for the European Train Control System with hybrid train detection. *CoRR*, 2024. `arXiv:2308.02572`.

5 Stefan Engels, Tom Peham, and Robert Wille. A symbolic design method for ETCS Hybrid Level 3 at different degrees of accuracy. In *23rd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS)*, 2023. `doi:10.4230/OASICS.ATMOS.2023.6`.

6 Stefan Engels and Robert Wille. Comparing lazy constraint selection strategies in train routing with moving block control. *CoRR*, 2024. `arXiv:2405.18977`.

7 D.C. Gill and C.J. Goodman. Computer-based optimisation techniques for mass transit railway signalling design. *IEE Proceedings B Electric Power Applications*, 139(3), 1992. `doi:10.1049/ip-b.1992.0031`.

8 Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. URL: `https://www.gurobi.com`.

9 Torsten Klug, Markus Reuther, and Thomas Schlechte. Does laziness pay off? - a lazy-constraint approach to timetabling. In *22nd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS)*, 2022. `doi:10.4230/OASIcs.ATMOS.2022.11`.

10 Jörn Pachl. *Railway Signalling Principles: Edition 2.0*. TU Braunschweig, 2021. `doi:10.24355/dbbs.084-202110181429-0`.

11 Tom Peham, Judith Przigoda, Nils Przigoda, and Robert Wille. Optimal railway routing using virtual subsections. In *Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification*. Springer, 2022. `doi:10.1007/978-3-031-05814-1_5`.

12 Thomas Schlechte, Ralf Borndörfer, Jonas Denißen, Simon Heller, Torsten Klug, Michael Küpper, Niels Lindner, Markus Reuther, Andreas Söhlke, and William Steadman. Timetable optimization for a moving block system. *Journal of Rail Transport Planning & Management*, 22, 2022. `doi:10.1016/j.jrtpm.2022.100315`.

13 Valeria Vignali, Federico Cuppi, Claudio Lantieri, Nicola Dimola, Tomaso Galasso, and Luca Rapagnà. A methodology for the design of sections block length on ETCS L2 railway networks. *Journal of Rail Transport Planning & Management*, 13, 2020. `doi:10.1016/j.jrtpm.2019.100160`.

14 Robert Wille, Tom Peham, Judith Przigoda, and Nils Przigoda. Towards automatic design and verification for Level 3 of the European Train Control System. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021. `doi:10.23919/date51398.2021.9473935`.

A Bayesian Rolling Horizon Approach for Rolling Stock Rotation Planning with Predictive Maintenance

Felix Prause[1]
Zuse Institute Berlin, Germany

Ralf Borndörfer
Zuse Institute Berlin, Germany

Abstract

We consider the rolling stock rotation planning problem with predictive maintenance (RSRP-PdM), where a timetable given by a set of trips must be operated by a fleet of vehicles. Here, the health states of the vehicles are assumed to be random variables, and their maintenance schedule should be planned based on their predicted failure probabilities. Utilizing the Bayesian update step of the Kalman filter, we develop a rolling horizon approach for RSRP-PdM, in which the predicted health state distributions are updated as new data become available. This approach reduces the uncertainty of the health states and thus improves the decision-making basis for maintenance planning. To solve the instances, we employ a local neighborhood search, which is a modification of a heuristic for RSRP-PdM, and demonstrate its effectiveness. Using this solution algorithm, the presented approach is compared with the results of common maintenance strategies on test instances derived from real-world timetables. The obtained results show the benefits of the rolling horizon approach.

2012 ACM Subject Classification Applied computing → Transportation; Mathematics of computing → Bayesian computation; Mathematics of computing → Mathematical optimization

Keywords and phrases Rolling stock rotation planning, Predictive maintenance, Rolling horizon approach, Bayesian inference, Local neighborhood search

Digital Object Identifier 10.4230/OASIcs.ATMOS.2024.13

Funding This work was supported by the innovation funding program ProFIT (grant no. 10186617), funded by the State of Berlin and co-funded by the European Union.

1 Introduction

Rail transport is one of the most positive modes of transport concerning environmental friendliness and sustainability. Its volume is likely to increase further in the future. This leads to an increased complexity in the planning of vehicle rotations and results in more challenging scenarios, particularly with respect to maintenance scheduling.

A maintenance strategy that has become increasingly important in recent years is predictive maintenance (PdM). One reason is the availability of sensors and the ability to analyze the data they generate using machine learning. In addition, PdM has obvious advantages in terms of economic and ecological factors. These advantages are based on the fact that the costs for spare parts are lower if the currently installed components are used until the end of their service life. As this also minimizes the number of spare parts used, the environmental impact is likewise reduced.

To combine the advantages of rail transport with those of PdM, it is necessary to develop approaches that integrate predictive maintenance planning into the optimization of rolling stock rotations.

[1] Corresponding author.

24th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2024).
Editors: Paul C. Bouman and Spyros C. Kontogiannis; Article No. 13; pp. 13:1–13:19
OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

One problem that arises in the application of PdM strategies is that the considered health states are usually unobservable quantities that have to be derived from measurements using some model. There are various steps in this process that introduce uncertainty into the obtained states. For example, measurement errors may occur when recording the observable quantities, or the employed model may be imprecise or only approximate. Since the states are subsequently projected into the future to serve as a basis for maintenance decisions, and the exact operating conditions cannot be known at that point in time, the uncertainty of the health states increases the further they are projected into the future. This uncertainty must be taken into account and addressed when optimizing the vehicle rotations.

Contribution

We consider the rolling stock rotation planning problem with predictive maintenance (RSRP-PdM), as presented in [31], and propose a rolling horizon approach that reduces the uncertainty of the health states and their future predictions. For this purpose, the RSRP-PdM is solved and the determined vehicle rotations are partially operated until measurements of the health states become available, for example, by analyzing sensor data collected during operation when the vehicles are parked overnight. Using these observations of the health conditions, the current predictions of the states are then updated by Bayesian inference. This reduces the variance of the health states and results in a subsequent RSRP-PdM instance.

Furthermore, we introduce a local neighborhood search, which is a modification of the heuristic presented in our earlier paper [29], to solve the occurring RSRP-PdM instances.

Outline

The article is structured as follows: First, in Section 2, we review and discuss the literature on the two arising tasks, i.e., the rolling stock rotation planning problem (RSRP) and predictive maintenance (PdM). Next, the problem formulation of RSRP-PdM, as stated in [31], is reproduced in Section 3. In Section 4, we describe the utilized Bayesian inference procedure. We provide a description of the rolling horizon approach in Section 5 and present a heuristic that extends the algorithm proposed in [29]. Section 6 then introduces the considered maintenance strategies, which are compared with each other in the subsequent computational experiments. Finally, we draw a conclusion on the obtained results in Section 7.

2 Related Work

Both aforementioned topics, i.e., PdM as well as RSRP, have already been thoroughly described and studied in the literature. In the following, we provide a brief overview of articles dealing with these two subjects.

The Rolling Stock Rotation Planning Problem (RSRP)

In RSRP, we are given a fleet of vehicles and a timetable whose trips must be operated. Furthermore, maintenance requirements are defined that the vehicles have to fulfill. The task is then to determine rotations for the rolling stock that operate all trips, satisfy the specified maintenance conditions, and have minimum costs. The articles addressing RSRP can be categorized according to the following three aspects:

- Regarding the model used to represent the vehicle rotations.
- According to the applied solution approach.
- Based on the employed maintenance strategy.

For a survey of the RSRP literature, we refer to [32]. An extensive comparison of different articles with an emphasis on additionally considered constraints can be found in [31].

The RSRP is usually modeled by space-time graphs, where the nodes correspond to events specified by a location and a time point, and the arcs represent the different actions of the vehicles, e.g., [8, 25, 38]. Then, there is the sequence model, in which the nodes depict the trips that need to be operated and the arcs indicate whether two tasks can be conducted in succession, see [6]. Next, we have the hypergraph model utilized by [4, 18, 32]. This model is a generalization of the space-time graph in which the hyperarcs represent vehicle compositions and their orientation during operation. Finally, there is the state-expanded event-graph, see [29, 31]. This is a space-time graph that is extended by additional dimensions to implicitly track the resource flow that enforces the maintenance constraints. In contrast to the previously mentioned approaches, this model allows for non-linear degradation functions, which is of particular relevance as the wear of mechanical components often exhibits such a behavior.

In all these models, the solutions to RSRP are given by flows that cover the trip arcs or nodes sufficiently often and satisfy the additional maintenance or capacity constraints. These induced flow problems are then solved by using branch and bound [8], the direct application of mixed-integer linear programs [17, 20, 21, 34, 39], column generation [2, 4, 32], branch and price [14, 25, 32], or the utilization of heuristics [5, 6, 18, 38].

The literature primarily employs preventive maintenance strategies. These include time-based maintenance [2, 4, 8, 14, 32] and distance-based maintenance [2, 4, 17, 21, 25, 32]. Recently, also condition-based and predictive maintenance regimes have been introduced for rail transport. Here, the maintenance decisions are either based on the vehicle states [5, 20], a classification of the vehicle conditions into degradation stages [39], or the remaining useful life (RUL) of the vehicles [34]. Finally, there are solution approaches that schedule the maintenance based on the predicted failure probability of the rolling stock [29, 31].

Predictive Maintenance (PdM)

The literature regarding PdM generally focuses on the prediction of the RUL or indices representing the future health conditions of the vehicles. We refer to [22] for different concepts of health indices. These indices usually do not consider the vehicles themselves, but rather the mechanical, electrical, or hydraulic components that are installed in them. The employed approaches are often distinguished into data-driven and model- or physics-based ones, see [1]. In addition, there are hybrid methods that combine both. A comprehensive literature review on PdM can be found in [13].

Data-driven models usually rely on machine learning techniques like classical neural networks (NNs), recurrent neural networks such as long short-term memory (LSTM) networks [10], support vector machines (SVMs) and decision trees [23], or deep learning [11]. For a survey of data-driven approaches applied in the railroad sector, see [9].

Model-based approaches, on the other hand, are based on some assumptions regarding the degradation process and often utilize Bayesian updating procedures [7, 16, 27].

Finally, there are hybrid approaches that combine model-based methods with machine learning. These include, for example, relevance vector machines, i.e., Bayesian variants of SVMs [35], Bayesian inference applied to the output of NNs [15, 26], or the assumption of a piecewise linear degradation behavior, followed by the subsequent combination of the outputs of multiple NNs using a Kalman filter [24]. In addition, a concept for extending deep learning models to Bayesian NNs is presented in [28], which enables the NNs to determine probability distributions for the predicted RUL.

Note that the hybrid approaches and the Bayesian methods have the advantage of obtaining a probability distribution for the RUL that captures the uncertainty of the prediction.

3 Problem Description

We consider the rolling stock rotation planning problem with predictive maintenance (RSRP-PdM), as presented in [31], and recall its description in the following. Suppose we are given a train timetable $\mathcal{T}$ consisting of various trips that need to be operated, and we have a homogeneous fleet of vehicles $\mathcal{V}$ at our disposal to conduct them. Each trip $t \in \mathcal{T}$ features a departure and an arrival location, i.e., $l_t^d, l_t^a \in \mathcal{L}$, as well as a departure and an arrival time, i.e., $k_t^d, k_t^a \in \mathcal{K}$. Here, $\mathcal{L}$ is the set of locations and $\mathcal{K}$ is the time horizon, which consists of a finite set of time points. Furthermore, we associate an integer $n_t \in \mathbb{Z}_{>0}$ with each trip, which indicates how many vehicles are required to operate t.

The task of RSRP-PdM is not only to find a feasible sequence of trips for each vehicle, i.e., a set of trips in which each pair of time-consecutive trips can be operated in succession, but also to schedule the maintenance of the vehicles. The maintenance actions can be carried out at the maintenance locations $\mathcal{L}_M \subseteq \mathcal{L}$ and should be based on the predicted health states of the vehicles. These health states are considered to be random variables since they cannot be measured directly and are thus prone to measurement, determination, and prediction errors. In addition, they are assumed to be normally distributed, i.e., $H_{v,k} \sim \mathcal{N}(\mu, \sigma^2)$ for some $\mu \in [0,1]$ and $\sigma^2 > 0$. Hence, all health states are distributed by the family of normal distributions with parameter space $\Theta = [0,1] \times \mathbb{R}_{>0}$ and can thus be characterized by their corresponding parameters $\theta \in \Theta$. In this sense, we assume that the initial state of each vehicle $v \in \mathcal{V}$ is given by an initial parameter $\theta_{v,0} \in \Theta$ that determines $H_{v,0}$. Note that a health value of one corresponds to a condition that is as good as new, while a value of zero or less signifies that the vehicle has a breakdown. The failure probability of a vehicle at a certain time is therefore given by

$$\mathbb{P}_f(v,k) := \mathbb{P}[H_{v,k} \leq 0] = \int_{-\infty}^{0} \frac{\exp\left(-\frac{\mu_{v,k}^2}{2\sigma_{v,k}^2}\right)}{\sqrt{2\pi\sigma_{v,k}^2}}\, dx = \frac{1}{2}\left(1 + \operatorname{erf}\left(\frac{-\mu_{v,k}}{\sqrt{2\sigma_{v,k}^2}}\right)\right),$$

where erf is the Gauss error function.

The deterioration of the vehicles and their maintenance is then described by modifying the parameters that represent their health states. Therefore, we associate a degradation function $\Delta_t : \Theta \to \Theta$ with each of the trips $t \in \mathcal{T}$, whose application describes the wear that occurs during the operation of t. Let $\tau_t := k_t^a - k_t^d$ be the duration of t, then the parameters of v after conducting t are determined by $\theta_{v,k+\tau_t} = \Delta_t(\theta_{v,k})$. To obtain a reasonable deterioration behavior, we further demand $\mu_{v,k+\tau_t} \leq \mu_{v,k}$ and $\sigma_{v,k+\tau_t}^2 \geq \sigma_{v,k}^2$ for $(\mu_{v,k+\tau_t}, \sigma_{v,k+\tau_t}^2) = \Delta_t(\mu_{v,k}, \sigma_{v,k}^2)$, i.e., the mean of the health state decreases, while the uncertainty about the condition grows. Similarly, we can associate degradation functions with the other activities of the vehicles such as waiting or deadheading, which might depend on the duration or the mileage of the corresponding task. Maintenance is also described by a wear function, which does not cause a deterioration of the condition, but resets the parameters of the health state to a certain value $\theta_m \in \Theta$.

We further assume that we occasionally receive measurements $y_{v,k}$ of the true health value of v at time k that exhibit noise originating from the measuring process. This noise is supposed to be normally distributed around zero with known variance $\sigma_y^2 > 0$.

Combining these notions, the task of RSRP-PdM is to determine a feasible assignment of the trips to the vehicles such that each trip is operated by the required number of vehicles. The objective here is to find an assignment of minimum total cost that takes the potential failure costs into account, i.e., the product of the predicted failure probability during the operation of the trips and the costs associated with the breakdown of a vehicle. Finally, we require that the rotations must be balanced, i.e., the number of vehicles located at each destination at the beginning and at the end of the time horizon have to coincide. This constraint is important as it gives rise to schedules that can be repeated periodically.

4 Bayesian Inference

In RSRP-PdM, as described in Section 3, the health states of the vehicles and their predictions are assumed to be random variables to reflect their uncertainty. Since these random variables are distributed by members of a family of probability distributions, their associated probability density functions (PDFs) can be characterized by their parameters, and the degradation functions describe how these parameters change. The deterioration process can therefore be understood as a dynamical system in which the parameters of the vehicle conditions represent the system states, and the degradation functions define the state transitions.

One problem that arises in practice is that the degradation functions are also subject to uncertainty since they can only be derived from historical data, and the actual operating conditions are unknown at the time of planning. If this uncertainty is factored in, the variance, and thus the uncertainty, of the predicted health states increases the further into the future they are projected. However, these predicted conditions form the basis for maintenance planning. Therefore, it should be attempted to reduce their variance to obtain more accurate estimates of the actual health states. This can be achieved by updating the predicted states with measurements, which is a filtering problem, see [36]. Using standard terminology, we utilize the *rule of Bayes* for these updates, see, for example, [36].

▶ **Theorem 1** (Rule of Bayes [36]). *Let θ and y be random variables representing a parameter estimate and a measurement, then it holds*

$$\mathbb{P}[\theta \mid y] = \frac{\mathbb{P}[y \mid \theta] \cdot \mathbb{P}[\theta]}{\mathbb{P}[y]} \propto \mathbb{P}[y \mid \theta] \cdot \mathbb{P}[\theta].$$

Here, $\mathbb{P}[\theta]$ is the *prior belief* about θ before obtaining measurement y, $\mathbb{P}[y \mid \theta]$ describes the *likelihood*, i.e., the relationship between the true state and y, which represents the measurement error. $\mathbb{P}[\theta \mid y]$ is the belief about θ after the information about y is incorporated, i.e., the *posterior belief*, and $\mathbb{P}[y]$ is the *marginal probability*, which can be interpreted as a normalization constant ensuring that $\mathbb{P}[\theta \mid y]$ has an integral of one. Furthermore, $f(x) \propto g(x)$ signifies that $f(x)$ is proportional to $g(x)$, i.e., there exists a constant $c \in \mathbb{R}$ such that $f(x) = c \cdot g(x)$ for all $x \in \mathbb{R}$. For further details, we refer to [36].

We now transfer the notions of Theorem 1 to the situation in RSRP-PdM. Recall that we assumed in Section 3 that the health states are normally distributed with a mean between zero and one. Suppose we are given a vehicle v at time k_0 with health state $H_{v,k_0} \sim \mathcal{N}(\mu_{k_0}, \sigma_{k_0}^2)$ that operates some services $\mathcal{S} = \{s_1, \dots, s_n\}$, i.e., a set consisting of trips, waiting times, deadhead trips, and maintenance actions in their chronological order. Then, a prediction of the health state of v after the operation of $\mathcal{S}$ can be determined by applying the degradation functions of the services in $\mathcal{S}$ to the parameters characterizing H_{v,k_0}. If we set $\Delta_{\mathcal{S}} := \Delta_{s_n} \circ \dots \circ \Delta_{s_1}$ and $k := k_0 + \tau_{s_1} + \dots + \tau_{s_n}$, the predicted state is therefore $\hat{H}_{v,k} \sim \mathcal{N}(\hat{\mu}_k, \hat{\sigma}_k^2)$ with $(\hat{\mu}_k, \hat{\sigma}_k^2) = \Delta_{\mathcal{S}}(\mu_{k_0}, \sigma_{k_0}^2)$.

Here, $\hat{H}_{v,k}$ represents the current belief about the true health state $H_{v,k}$. If we now obtain a measurement y of the true health state with a measuring error whose variance is specified by $\sigma_y^2 > 0$, then we have $y \mid \hat{H}_{v,k} \sim \mathcal{N}(\hat{H}_{v,k}, \sigma_y^2)$. Thus, we can apply the rule of Bayes to obtain $H_{v,k} = \hat{H}_{v,k} \mid y$. Since the considered random variables are all normally distributed, this inference corresponds precisely to the update step of the Kalman filter, as described in [36].

▶ **Definition 2** (Kalman Filter Update Step). *Let $\hat{\mu} \in \mathbb{R}^n$ be the predicted mean of the estimated state and $\hat{P} \in \mathbb{R}^{n\times n}$ the corresponding predicted covariance matrix. Furthermore, let $H \in \mathbb{R}^{m\times n}$ be the measurement model, $y \in \mathbb{R}^m$ a measurement of the true state and $R \in \mathbb{R}^{m\times m}$ the corresponding covariance matrix of the measurement. Then, the* Kalman filter update step *is defined as follows:*

$$\begin{aligned} K &= \hat{P}H^T \left(H\hat{P}H^T + R\right)^{-1} \\ \mu &= \hat{\mu} + K\left(y - H\hat{\mu}\right) \\ P &= \left(I_n - KH\right)\hat{P}, \end{aligned}$$

where $K \in \mathbb{R}^{n\times m}$ is the Kalman gain, *$\mu \in \mathbb{R}^n$ is the updated mean of the estimated state, $P \in \mathbb{R}^{n\times n}$ is the corresponding updated covariance matrix, and $I_n \in \mathbb{R}^{n\times n}$ is the identity matrix of size n.*

Considering Definition 2 for the one-dimensional case and assuming that the measurement is a direct observation of the true state, i.e., $H = 1$, we obtain the following corollary:

▶ **Corollary 3.** *Let $\mu_\theta \in \mathbb{R}$ be the predicted mean of the estimated state and $\sigma_\theta^2 > 0$ the corresponding predicted variance. Furthermore, let $\mu_y \in \mathbb{R}$ be a direct measurement of the true state and $\sigma_y^2 > 0$ the corresponding variance of the measurement noise. Then, applying the Kalman filter update step yields the updated state estimate*

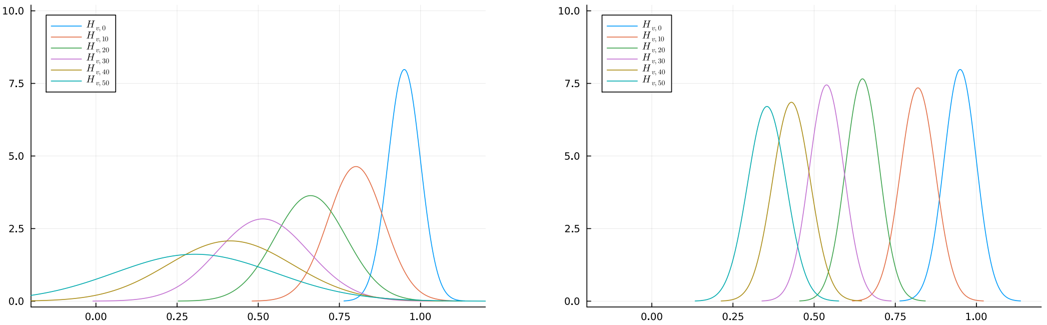
$$X \sim \mathcal{N}\left(\frac{\mu_\theta\sigma_y^2 + \mu_y\sigma_\theta^2}{\sigma_\theta^2 + \sigma_y^2}, \frac{\sigma_\theta^2\sigma_y^2}{\sigma_\theta^2 + \sigma_y^2}\right).$$

The inferred health states can therefore be directly determined by applying Corollary 3. Moreover, they are also normally distributed and thus members of the considered family of probability distributions.

An example illustrating the impact of inference is given in Figure 1. The two graphs show the development of a vehicle's predicted/updated health state over time. The corresponding PDFs are given from right to left and show the state after the operation of ten trips in each case, starting with the initial health $H_{v,0}$ on the far right. It can be observed that the variance in the scenario without inference continues to increase, while it remains in the same order of magnitude if the states are updated with measurements after every ten trips. As a result, $H_{v,50}$ has a high probability of taking values between 0.25 and 0.45 in the second case, whereas in the first case, it contains essentially no information since its variance is too large.

Application to Non-normally Distributed Health States

However, the health states do not necessarily have to be represented by normally distributed random variables. Other possible models could, for example, be based on families of probability distributions that are used in reliability theory, such as the family of Weibull or gamma distributions, see, e.g., [12].



(a) PDF of the predicted health state over time, without state updates.

(b) PDF of the health state over time, using inference after every tenth trip.

Figure 1 Comparison of the health state PDF development over time, with and without inference.

Furthermore, it would also be conceivable that the states are distributed by discrete distributions or PDFs that do not belong to any common family of distributions. In these cases, we have to choose a family of distributions whose members fit the given data as well as possible since the employed approach is based on the assumption that the PDFs of the random variables representing the health states all belong to a parametric family.

Suppose now that we have determined such a family of distributions for modeling the health states and are able to obtain measurements of the true states, which are either given by point estimates with a certain measuring error or described by a PDF. If we then apply the rule of Bayes to infer the posterior of the health state after incorporating the measurement, as described above, the resulting PDF does not necessarily have to be a member of the selected family. It may even be that it does not belong to any common family of distributions.

In these cases, the posterior of the state would have to be approximated by a member of the considered family of distributions. This would, for example, be possible by employing a Markov chain Monte Carlo algorithm to sample from the posterior distribution of the health state. Subsequently, we determine the PDF of the family that best fits the obtained data w.r.t. some statistical distance. Another option would be the utilization of a variational Bayesian method, see, e.g., [37].

5 Solution Approach

In this section, we introduce the rolling horizon approach. This algorithm determines a solution for RSRP-PdM by sequentially generating and solving sub-instances. Here, the instances occurring in each iteration are solved by a heuristic, which is also described below.

5.1 The Rolling Horizon Approach

The idea of the rolling horizon approach is the following: Suppose we have a solution for RSRP-PdM where the expected deterioration caused by the trips is only known approximately. Then, the maintenance services of the vehicles are planned entirely based on predictions of the health states, whose uncertainty increases the further they are projected into the future. However, if it is possible to occasionally obtain measurements of the health states and incorporate them into the planning of the rotations, these can be adjusted such that the vehicles are assigned to trips that better match their conditions. In addition, maintenance needs can be better estimated, and the vehicles can be serviced at more appropriate times.

This can be accomplished by first solving the initial RSRP-PdM instance (`solveInstance`) and operating the resulting vehicle rotations until measurements y of the health states become available at a certain time k_y (`operateRotations`). Subsequently, a new problem instance is created by restricting the timetable of the considered instance to trips whose departure time is greater than or equal to k_y (`restrictTrips`). The positions of the vehicles are then updated by assigning them the location at which they are at time k_y or at which they terminate the operation they are conducting at that time (`updatePositions`). In addition, the predicted health states of the vehicles are updated with y using Bayesian inference, as described in Section 4 (`inferStates`). This procedure is outlined in Algorithm 1 and results in a new RSRP-PdM instance, which is then considered in the next iteration of the approach. The algorithm stops when all trips of the original instance have been completed.

Algorithm 1 One iteration of the rolling horizon approach.

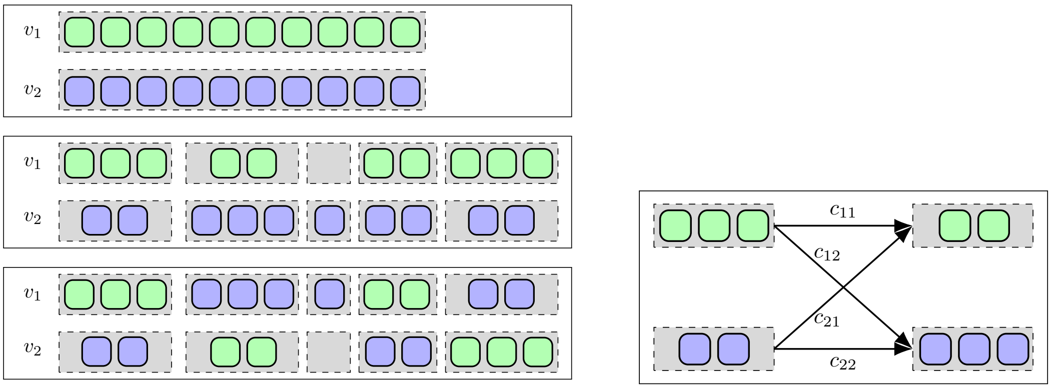
```
Input  : RSRP-PdM instance $\mathcal{I}$, measurements $y$ of the health states at time $k_y$
Output : Updated RSRP-PdM instance for the remaining time
$s \leftarrow$ solveInstance($\mathcal{I}$)
operateRotations($s, k_y$)
$\mathcal{I}' \leftarrow$ restrictTrips($\mathcal{I}, k_y$)
$\mathcal{I}' \leftarrow$ updatePositions($\mathcal{I}', s, k_y$)
$\mathcal{I}' \leftarrow$ inferStates($\mathcal{I}', y$)
return $\mathcal{I}'$
```

5.2 A Local Neighborhood Search That Considers Transition Costs

To solve the occurring RSRP-PdM instances, we utilize a modified version of the multi-swap heuristic proposed in [29]. This algorithm initially solves the underlying RSRP of the instance using an integer linear program that neglects the maintenance constraints. Afterwards, maintenance is scheduled in a second step by determining a shortest path in the state-expanded event-graph (SEEG). Then, a local neighborhood search is employed to improve the rotations of the current solution, where maintenance planning is again done by searching for a shortest path in the SEEG.

This local neighborhood search works as follows: Given two vehicle rotations, the first step is to determine the possible swap positions, i.e., the times at which both vehicles can reach and operate the next trip of the other. This groups the trips of the rotations into sets that can be exchanged without violating the feasibility of the corresponding vehicle schedules. Subsequently, the generated subsets of trips are randomly swapped to obtain new rotations. An example of this procedure is shown in Figure 2a. Here, the rotations of vehicles v_1 and v_2 (top) are first grouped into subsets of interchangeable trips (middle). Afterwards, some of these related subsets are swapped to create two new vehicle schedules (bottom).

In [29], the swapping decisions are sampled from a discrete uniform distribution $\mathcal{U}\{0,1\}$, i.e., the swaps are performed with a probability of one-half. However, some swaps are more beneficial than others in terms of costs, and their selection may accelerate the process of finding good solutions. Therefore, the transition costs of the vehicles should be included when determining the exchange probabilities. These costs are referred to as $c_{ij} \in \mathbb{R}_{\geq 0}$, for $i, j \in \{1, 2\}$, and are associated with the waiting times and deadhead trips that are necessary for vehicle v_i to reach the next trip of vehicle v_j. For example, consider the scenario depicted in Figure 2b with costs $c_{11} = c_{22} = 1$ and $c_{12} = c_{21} = 10$. If we disregard maintenance



(a) The various stages of the local neighborhood search.

(b) The costs used in the weighted variant of the swapping procedure.

Figure 2 Visualization of the stages of the employed heuristic (a) and of the transition costs of the vehicles between trip subsets of the second stage (b).

decisions, a swap of the second pair of trip subsets would lead to an increase in transition costs. Therefore, it would not be advisable to swap with a probability of one-half at this position. For this reason, we use a probability of

$$\mathbb{P}_s = \begin{cases} \frac{c_{11}+c_{22}}{c_{11}+c_{12}+c_{21}+c_{22}} & \text{if } c_{11}+c_{12}+c_{21}+c_{22} \neq 0 \\ \frac{1}{2} & \text{else} \end{cases}$$

for sampling the swapping decisions. However, to ensure a certain degree of exploration, the $\mathbb{P}_s$ were finally rounded to values in $[0.05, 0.95]$. The swapping decisions for each pair of trip subsets are then made in chronological order with the associated probability $\mathbb{P}_s$. Here, we set $\mathbb{P}_s := 1 - \mathbb{P}_s$ if the previous decision was a swap to account for the resulting exchange of c_{11} and c_{12}, as well as of c_{22} and c_{21}.

A comparison of the solutions obtained with the multi-swap heuristic using this swapping procedure with those of the algorithm from [29] can be found in Appendix A. The results demonstrate the effectiveness of the modified swapping approach.

6 Computational Results

In this section, we examine the results of the Bayesian rolling horizon approach proposed in Section 5.1. For this purpose, we compare its solutions with those of two other maintenance strategies, namely predictive maintenance without Bayesian inference and preventive maintenance. To solve the occurring RSRP and RSRP-PdM instances, we apply the *weighted multi-swap heuristic* presented in Section 5.2 and use scenarios derived from the instances given in [30] for testing.

Computational Setup

The data structures and algorithms were implemented in Julia v1.9.4 [3], and Gurobi v10.0.2 [19] was employed to solve the integer programs that are used to find initial rotations for the heuristics. The computations were conducted on a computer with Intel(R) Xeon(R) Gold 6342 @ 2.80GHz CPUs, eight cores, and 64GB of RAM. Finally, all approaches had a time limit of seven hours, with the rolling horizon approach using one hour to solve the RSRP-PdM instance of each day of the given week.

6.1 Test Instances

The considered test scenarios, used for the evaluation and comparison of the maintenance regimes, are based on the instances T1 – T6 constructed in [30]. Each of them has an individual timetable, which needs to be completed within one week, and contains the information about the available fleet, i.e., the operating costs of the vehicles and their initial positions. In addition, the distances between the considered locations and the costs associated with trips, waiting times, deadheading, and maintenance are indicated. The costs for vehicle breakdowns are also defined.

The components that specify the conditions of the vehicles are their doors, and it is assumed that they can undergo 1,500 cycles before failing. This assumption is based on the real-world data used in [33] and influences the number of required maintenance services. Furthermore, the expected deterioration caused by the operation of each trip, i.e., the number of expected cycles, is given by the mean and variance of a normal distribution derived from the passenger volume at the served stations, see [30]. Note that the vehicle conditions in the conducted computations are assumed to deteriorate only during the operation of the trips. In the original instances, a non-linear degradation behavior is considered, whereas we assume a linear one, as this improves the comparability of the obtained solutions.

From each of these instances, we derive ten scenarios by sampling the initial states of the vehicles, i.e., the number of already performed cycles, from the discrete uniform distribution $\mathcal{U}\{0, 1200\}$. In addition, the number of cycles that actually arise during each trip is sampled from the corresponding normal distribution. These values are later used to determine whether vehicle failures occur during the operation of the obtained schedules and to derive the measurements utilized in the Bayesian approach. This gives rise to the various scenarios T1-01 – T6-10, which are used for the computations below.

Furthermore, we assume that, if the states are considered as random variables, the initial states of the vehicles, the conditions after maintenance, and the measurements each have a variance of 25, i.e., the corresponding values are accurate to within ten cycles with a probability of 90%. Finally, for the predictive approaches, we apply a transformation that converts the number of cycles into a health value between zero and one. Here, zero cycles correspond to a new condition, i.e., a value of one, while 1,500 cycles correspond to a vehicle failure and thus a value of zero. The variances were also adjusted accordingly.

6.2 Compared Maintenance Strategies

Now, we describe the maintenance strategies considered in the computational experiments. Since we assume a linear degradation behavior, one-dimensional quantities simply have to be added. However, if the states and occurring cycles of the trips are considered as random variables, the convolution of their PDFs must be determined. Nevertheless, since both are presumed to be normally distributed, this convolution can be calculated by summing the mean values and the variances of the corresponding PDFs.

Preventive Maintenance

The first considered maintenance strategy is preventive maintenance (PM), which represents the status quo of maintenance planning. In this approach, one-dimensional quantities such as the traveled distance or the elapsed time are usually considered, and maintenance is performed before the values exceed a certain threshold.

Applied to the considered scenarios, we assume that maintenance decisions are based on the number of operated cycles, but only the mean values of the distributions characterizing the deterioration due to the trips are used. Since the vehicles are assumed to complete 1,500 cycles before failing, we must first define a suitable threshold to avoid vehicle breakdowns. For this purpose, we examine the preventive strategy with safety margins of 5% and 10%.

Based on the 3σ *rule*, i.e., the fact that approximately 99.7% of the values in a normal distribution lie within an interval of three standard deviations around the mean, we require that $\mu + 3\sigma \leq 1,500$ holds for all parameters that the predicted health state of a vehicle could possess before maintenance. Let v be a vehicle in new condition, i.e., with parameters $(\mu, \sigma^2) = (0, 25)$. Then, we apply the degradation functions of randomly selected trips to v until the mean value of its predicted state is 1,350 and 1,425, respectively. With 50,000 repetitions, this results in a maximum variance of 750 or 800. Thus, a margin of 5% does not fulfill the property required above and might therefore lead to failures. Hence, we use a threshold of 1,350 cycles, i.e., a safety margin of 10%, for PM.

Predictive Maintenance

The next strategy used is the direct application of predictive maintenance (PdM), as described in [29, 31]. Here, maintenance planning is based on the predicted failure probabilities of the vehicles. However, the health states are not updated. Since the degradation behavior is assumed to be linear, the deterioration of a vehicle v due to a trip t is expressed by the convolution of the PDF of the random variable representing the health state of v with the PDF of the random variable describing the expected number of cycles occurring during the operation of t. As previously mentioned, the initial vehicle states and the conditions after maintenance are assumed to have a variance of 25.

Predictive Maintenance with Bayesian Inference

Finally, we consider the maintenance strategy developed in this article, namely predictive maintenance with Bayesian inference (PdM-B). For this, we utilize the predictive maintenance regime PdM described above but additionally assume that it is possible to receive measurements of the health states when the vehicles are parked overnight. As explained in Section 4, we assume that these measurements are normally distributed and have a fixed variance $\sigma_y^2 = 25$, which reflects the measuring error. After obtaining the measurements of the states, we apply Algorithm 1, using Corollary 3 to update the predicted vehicle states and reduce their variance. This yields a new RSRP-PdM instance, which is subsequently solved.

6.3 Results

The results of the computations are summarized in Table 1 and show the number of maintenance services and the total costs of the solutions after averaging the scenarios derived from each instance. The best results for each instance are marked in bold. A more detailed itemization of the costs by type can be found in Tables 3–5 in Appendix B. Here, only the actually incurred costs are taken into account, i.e., the expected failure costs, which are considered in the predictive maintenance strategies for maintenance planning, are ignored. In addition, all determined vehicle rotations were compared with the number of cycles that actually occurred, i.e., the number of cycles that were sampled for each trip when the scenarios were created. However, none of the generated solutions resulted in a vehicle failure.

Table 1 Number of maintenance services and total costs of the solutions generated by the considered maintenance strategies after averaging the scenarios of each instance.

Instance	Maintenance services			Total costs		
	PM	PdM	PdM-B	PM	PdM	PdM-B
T1	6.3	**5.8**	**5.8**	272,822	**271,781**	271,937
T2	**4.9**	**4.9**	**4.9**	462,219	466,564	**458,174**
T3	14.9	13.3	**13.2**	1,421,806	**1,411,300**	1,412,457
T4	6.0	5.5	**5.1**	212,015	212,069	**211,831**
T5	7.0	6.8	**6.2**	346,513	344,891	**342,976**
T6	21.1	18.8	**18.7**	**2,369,415**	2,377,673	2,375,013
Σ	60.2	55.1	**53.9**	5,084,790	5,084,278	**5,072,388**

A comparison of PM with PdM shows that both approaches achieve better results in terms of costs than the other for three of the six instances. Overall, however, they have almost the same total costs. The reason for this is that PdM reduces the number of maintenance services but does so at the expense of higher deadhead costs, compare Tables 3 and 4. Nevertheless, excluding instance T2, PdM performed fewer maintenance services for each instance and was able to reduce the total number of service actions by 5.1.

PdM-B, on the other hand, is able to reduce the number of conducted maintenance operations even further. To be precise, an average of 6.3 fewer maintenance operations are required compared to PM. Moreover, it can compensate for the disadvantage of PdM and reduces the deadhead costs to such an extent that it generates the vehicle schedules with the lowest total costs. It was thus able to achieve lower costs than PM for all instances except T6 and reduced the number of maintenance services for all instances except for T2. In particular, the number of service actions for instances with many vehicles, i.e., for T3 and T6, is reduced. Looking at the entire network, i.e., all instances combined, PdM-B was able to decrease the number of maintenance actions by 10.5%, while the costs are 0.24% lower. If the fixed costs, i.e., the costs required to operate the trips, are neglected, the overall cost advantage increases to 0.97%.

The cost differences, expressed as percentages, of PdM-B compared to PM and PdM are shown in Table 6. These differences range between -1.02% and 0.24% and between -1.8% and 0.08%, respectively. If the fixed costs are again excluded, the cost advantages for the individual instances increase to up to 5.72% and 8.05%, respectively.

The main cost benefits of PdM-B in comparison to PM are less maintenance (T1), lower deadhead costs (T2), or the combination of both (T3 and T5), see Tables 3 and 5. For T4, the costs for maintenance and deadhead trips are also decreased, but PdM-B deploys additional vehicles that may be necessary to absorb a more severe deterioration identified by the incorporation of the measurements. In addition, some maintenance can be spared by using these vehicles, which also reduces the costs. In the case of T6, PdM-B achieved solutions with higher costs than PM, which is due to the increased deadhead costs, although the number of maintenance tasks could be lowered.

Compared to PdM, PdM-B achieves lower costs for four of the six instances and lower costs overall, specifically 0.23% less. When the fixed costs are neglected, the benefit is 0.93%. In addition, the number of performed maintenance actions was reduced by a total of 1.2. The cost savings are again due to lower deadhead costs (T2 and T6) and to a combination of lower deadhead and maintenance costs (T4 and T5). For T1 and T3, the solutions generated by PdM-B have slightly higher costs than those generated by PdM. In the first case, this is due to the utilization of additional vehicles, which reduces deadhead costs, while in the second case, slightly higher deadhead costs are accepted to assign the vehicles to trips that better match their conditions.

7 Conclusion

In this article, we presented a rolling horizon approach for RSRP-PdM that incorporates health state measurements using Bayesian inference. We also extended the local neighborhood search from [29] to include transition costs when determining the swapping probabilities.

For this purpose, we first gave a literature review of the two topics RSRP and PdM. Then, we recalled the problem formulation of RSRP-PdM and introduced the notions that arise in the context of Bayesian inference. Subsequently, the Bayesian rolling horizon approach was introduced and we described the modified local neighborhood search. Finally, we conducted computational experiments with three different maintenance strategies, namely preventive and predictive maintenance, as well as predictive maintenance with Bayesian inference, and compared their solutions.

The results show that the iterative scheduling of the Bayesian approach is advantageous over both preventive and predictive maintenance without updating. Not only can the number of maintenance actions be reduced by 10.5% compared to the conventional strategy of preventive maintenance, but also the total costs of all instances combined are decreased by 0.24%, or by 0.97% if the fixed costs are excluded. In comparison to predictive maintenance without updating, the costs can likewise be reduced by 0.23% and 0.93% respectively due to fewer deadhead trips, while the number of maintenance operations is also slightly decreased. This demonstrates the effectiveness and benefits of the Bayesian rolling horizon approach.

In addition, we have shown that taking transition costs into account improves the performance of the multi-swap heuristic, both in terms of the solution value after a short and a long computation time.

References

1 Dawn An, Nam H. Kim, and Joo-Ho Choi. Practical options for selecting data-driven or physics-based prognostics algorithms with reviews. *Reliability Engineering & System Safety*, 133:223–236, 2015. doi:10.1016/j.ress.2014.09.014.

2 Javier Andrés, Luis Cadarso, and Ángel Marín. Maintenance scheduling in rolling stock circulations in rapid transit networks. *Transportation Research Procedia*, 10:524–533, 2015. doi:10.1016/j.trpro.2015.09.006.

3 Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B. Shah. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98, 2017. doi:10.1137/141000671.

4 Ralf Borndörfer, Markus Reuther, Thomas Schlechte, Kerstin Waas, and Steffen Weider. Integrated optimization of rolling stock rotations for intercity railways. *Transportation Science*, 50(3):863–877, 2016. doi:10.1287/trsc.2015.0633.

5 Omar Bougacha, Christophe Varnier, and Noureddine Zerhouni. Impact of decision horizon on post-prognostics maintenance and missions scheduling: a railways case study. *International Journal of Rail Transportation*, 10(4):516–546, 2022. doi:10.1080/23248378.2021.1940329.

6 Valentina Cacchiani, Alberto Caprara, and Paolo Toth. A fast heuristic algorithm for the train unit assignment problem. In *12th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012. doi:10.4230/OASIcs.ATMOS.2012.1.

7 Nan Chen and Kwok Leung Tsui. Condition monitoring and remaining useful life prediction using degradation signals: Revisited. *IiE Transactions*, 45(9):939–952, 2013. doi:10.1080/0740817X.2012.706376.

8 Jean-François Cordeau, François Soumis, and Jacques Desrosiers. Simultaneous assignment of locomotives and cars to passenger trains. *Operations research*, 49(4):531–548, 2001. doi:10.1287/opre.49.4.531.11226.

9 Narjes Davari, Bruno Veloso, Gustavo de Assis Costa, Pedro Mota Pereira, Rita P. Ribeiro, and João Gama. A survey on data-driven predictive maintenance for the railway industry. *Sensors*, 21(17):5739, 2021. doi:10.3390/s21175739.

10 Luigi de Simone, Enzo Caputo, Marcello Cinque, Antonio Galli, Vincenzo Moscato, Stefano Russo, Guido Cesaro, Vincenzo Criscuolo, and Giuseppe Giannini. Lstm-based failure prediction for railway rolling stock equipment. *Expert Systems with Applications*, 222:119767, 2023. doi:10.1016/j.eswa.2023.119767.

11 André Listou Ellefsen, Emil Bjørlykhaug, Vilmar Æsøy, Sergey Ushakov, and Houxiang Zhang. Remaining useful life predictions for turbofan engine degradation using semi-supervised deep architecture. *Reliability Engineering & System Safety*, 183:240–251, 2019. doi:10.1016/j.ress.2018.11.027.

12 Frank Emmert-Streib and Matthias Dehmer. Introduction to survival analysis in practice. *Machine Learning and Knowledge Extraction*, 1(3):1013–1038, 2019. doi:10.3390/make1030058.

13 Aurora Esteban, Amelia Zafra, and Sebastian Ventura. Data mining in predictive maintenance systems: A taxonomy and systematic review. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 12(5):e1471, 2022. doi:10.1002/widm.1471.

14 Yuan Gao, Jun Xia, Andrea D'Ariano, and Lixing Yang. Weekly rolling stock planning in chinese high-speed rail networks. *Transportation Research Part B: Methodological*, 158:295–322, 2022. doi:10.1016/j.trb.2022.02.005.

15 Nagi Z. Gebraeel and Mark A. Lawley. A neural network degradation model for computing and updating residual life distributions. *IEEE Transactions on Automation Science and Engineering*, 5(1):154–163, 2008. doi:10.1109/TASE.2007.910302.

16 Nagi Z. Gebraeel, Mark A. Lawley, Rong Li, and Jennifer K. Ryan. Residual-life distributions from component degradation signals: A bayesian approach. *IiE Transactions*, 37(6):543–557, 2005. doi:10.1080/07408170590929018.

17 Giovanni Luca Giacco, Andrea D'Ariano, and Dario Pacciarelli. Rolling stock rostering optimization under maintenance constraints. *Journal of Intelligent Transportation Systems*, 18(1):95–105, 2014. doi:10.1080/15472450.2013.801712.

18 Boris Grimm, Ralf Borndörfer, Markus Reuther, and Thomas Schlechte. A cut separation approach for the rolling stock rotation problem with vehicle maintenance. In *19th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2019)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/OASIcs.ATMOS.2019.1.

19 Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, Version 10.0.2. https://www.gurobi.com, 2024.

20 Nathalie Herr, Jean-Marc Nicod, Christophe Varnier, Noureddine Zerhouni, and Pierre Dersin. Predictive maintenance of moving systems. In *2017 Prognostics and System Health Management Conference (PHM-Harbin)*, pages 1–6. IEEE, 2017. doi:10.1109/PHM.2017.8079111.

21 Satoshi Kato, Naoto Fukumura, Susumu Morito, Koichi Goto, and Narumi Nakamura. A mixed integer linear programming approach to a rolling stock rostering problem with splitting and combining. In *RailNorrköping 2019. 8th International Conference on Railway Operations Modelling and Analysis (ICROMA), Norrköping, Sweden, June 17th–20th, 2019*, pages 548–564. Linköping University Electronic Press, 2019.

22 Yaguo Lei, Naipeng Li, Liang Guo, Ningbo Li, Tao Yan, and Jing Lin. Machinery health prognostics: A systematic review from data acquisition to rul prediction. *Mechanical systems and signal processing*, 104:799–834, 2018. doi:10.1016/j.ymssp.2017.11.016.

23 Hongfei Li, Dhaivat Parikh, Qing He, Buyue Qian, Zhiguo Li, Dongping Fang, and Arun Hampapur. Improving rail network velocity: A machine learning approach to predictive maintenance. *Transportation Research Part C: Emerging Technologies*, 45:17–26, 2014. doi:10.1016/j.trc.2014.04.013.

24 Pin Lim, Chi Keong Goh, Kay Chen Tan, and Partha Dutta. Estimation of remaining useful life based on switching kalman filter neural network ensemble. In *Annual Conference of the PHM Society*, volume 6(1), 2014. doi:10.36001/phmconf.2014.v6i1.2348.

25 Richard M. Lusby, Jørgen Thorlund Haahr, Jesper Larsen, and David Pisinger. A branch-and-price algorithm for railway rolling stock rescheduling. *Transportation Research Part B: Methodological*, 99:228–250, 2017. doi:10.1016/j.trb.2017.03.003.

26 Kursat Rasim Mestav, Jaime Luengo-Rozas, and Lang Tong. State estimation for unobservable distribution systems via deep neural networks. In *2018 IEEE Power & Energy Society General Meeting (PESGM)*, pages 1–5. IEEE, 2018. doi:10.1109/PESGM.2018.8586649.

27 Roberto Nappi, Gianluca Cutrera, Antonio Vigliotti, and Giuseppe Franzè. A predictive-based maintenance approach for rolling stocks vehicles. In *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 793–798. IEEE, 2020. doi:10.1109/ETFA46521.2020.9212183.

28 Weiwen Peng, Zhi-Sheng Ye, and Nan Chen. Bayesian deep-learning-based health prognostics toward prognostics uncertainty. *IEEE Transactions on Industrial Electronics*, 67(3):2283–2293, 2019. doi:10.1109/TIE.2019.2907440.

29 Felix Prause. A multi-swap heuristic for rolling stock rotation planning with predictive maintenance. In *Proceedings of the 11th International Network Optimization Conference (INOC), Dublin, Ireland, March 11-23, 2024*, pages 58–63, 2024. doi:10.48786/inoc.2024.11.

30 Felix Prause and Ralf Borndörfer. Construction of a test library for the rolling stock rotation problem with predictive maintenance. Technical Report 23-20, ZIB, Takustr. 7, 14195 Berlin, 2023.

31 Felix Prause, Ralf Borndörfer, Boris Grimm, and Alexander Tesch. Approximating rolling stock rotations with integrated predictive maintenance. *Journal of Rail Transport Planning & Management*, 30:100434, 2024. doi:10.1016/j.jrtpm.2024.100434.

32 Markus Reuther. *Mathematical optimization of rolling stock rotations*. PhD thesis, Technische Universität Berlin (Germany), 2017. doi:10.14279/depositonce-5865.

33 Rita P. Ribeiro, Pedro Pereira, and João Gama. Sequential anomalies: a study in the railway industry. *Machine Learning*, 105:127–153, 2016. doi:10.1007/s10994-016-5584-6.

34 Pegah Rokhforoz and Olga Fink. Hierarchical multi-agent predictive maintenance scheduling for trains using price-based approach. *Computers & Industrial Engineering*, 159:107475, 2021. doi:10.1016/j.cie.2021.107475.

35 Bhaskar Saha, Kai Goebel, Scott Poll, and Jon Christophersen. Prognostics methods for battery health monitoring using a bayesian framework. *IEEE Transactions on instrumentation and measurement*, 58(2):291–296, 2008. doi:10.1109/TIM.2008.2005965.

36 Simo Särkkä and Lennart Svensson. *Bayesian filtering and smoothing*, volume 17. Cambridge university press, 2023.

37 Shiliang Sun. A review of deterministic approximate inference techniques for bayesian machine learning. *Neural Computing and Applications*, 23:2039–2050, 2013. doi:10.1007/s00521-013-1445-4.

38 Per Thorlacius, Jesper Larsen, and Marco Laumanns. An integrated rolling stock planning model for the copenhagen suburban passenger railway. *Journal of Rail Transport Planning & Management*, 5(4):240–262, 2015. doi:10.1016/j.jrtpm.2015.11.001.

39 Meng-Ju Wu and Yung-Cheng Lai. Train-set assignment optimization with predictive maintenance. In *RailNorrköping 2019. 8th International Conference on Railway Operations Modelling and Analysis (ICROMA), Norrköping, Sweden, June 17th–20th, 2019*, pages 1131–1139. Linköping University Electronic Press, 2019.

A Computational Evaluation of the Weighted Swapping Procedure

In the following, we evaluate the effectiveness of the multi-swap heuristic utilizing the weighted swapping procedure presented in Section 5.2. We will refer to this approach as *weighted multi-swap* (WMS) and compare its results with those obtained by the algorithm from [29]. This uses a swapping probability of one-half and is therefore denoted as *equal probability multi-swap* (EPMS) in the following.

For testing, we use the instances of the preventive maintenance strategy described in Section 6. The detailed results are listed in Appendix A and show the values of the solutions obtained by EPMS and WMS after 360 seconds and after one hour of computation time. The best results for each instance are marked in bold.

We can observe that EPMS finds the best result after 360 seconds only eight times and that it was able to determine the best solution after one hour for only eight scenarios. The majority of these cases occur for instances T3 and T4, both of which are medium-sized scenarios in terms of the number of trips. In addition, there is a tie for the best solution for six and seven instances, respectively. WMS, on the other hand, could find the best solution after 360 seconds for 46 of the scenarios and achieved a better result after one hour for 45 of the 60 instances. These results show that the cost-oriented swapping used in WMS offers an advantage over the procedure employed in EPMS, both in terms of finding good solutions quickly and finding solutions with low costs.

Table 2 Results of EPMS and WMS after 360 seconds and after one hour of computation time.

Instance	EPMS after 360 s	WMS after 360 s	EPMS	WMS
T1-01	274,016	**272,090**	272,514	**272,090**
T1-02	276,858	**276,120**	276,544	**276,120**
T1-03	276,684	**275,607**	275,836	**275,607**
T1-04	272,514	**272,090**	272,514	**272,090**
T1-05	272,514	**272,090**	**272,090**	**272,090**
T1-06	**270,075**	**270,075**	**270,075**	**270,075**
T1-07	277,825	**276,544**	**276,544**	**276,544**
T1-08	272,514	**272,090**	**272,090**	**272,090**
T1-09	**272,090**	**272,090**	**272,090**	**272,090**
T1-10	**270,075**	**270,075**	**270,075**	**270,075**
T2-01	507,420	**494,036**	484,718	**468,615**
T2-02	511,781	**488,175**	496,993	**474,073**
T2-03	503,256	**492,784**	493,873	**471,772**
T2-04	486,411	**476,156**	480,048	**468,335**
T2-05	498,193	**466,304**	485,160	**459,522**
T2-06	500,491	**482,167**	480,559	**472,028**
T2-07	506,559	**485,151**	489,451	**474,629**
T2-08	496,731	**473,915**	484,462	**461,136**
T2-09	499,435	**475,127**	486,415	**461,121**
T2-10	500,269	**484,520**	481,735	**468,810**

Continued on next page

Table 2 – continued from previous page

Instance	EPMS after 360 s	WMS after 360 s	EPMS	WMS
T3-01	1,443,584	**1,417,645**	1,429,586	**1,415,815**
T3-02	1,466,915	**1,436,870**	1,459,579	**1,435,336**
T3-03	**1,471,162**	1,486,642	**1,446,601**	1,458,286
T3-04	1,465,143	**1,442,264**	1,448,476	**1,433,629**
T3-05	1,510,924	**1,441,830**	1,469,280	**1,432,776**
T3-06	**1,456,730**	1,475,282	**1,437,910**	1,456,083
T3-07	1,470,481	**1,443,980**	1,447,427	**1,441,852**
T3-08	1,430,814	**1,424,607**	**1,422,649**	1,424,329
T3-09	1,433,021	**1,419,417**	1,427,896	**1,418,864**
T3-10	1,449,332	**1,412,973**	1,427,362	**1,411,817**
T4-01	**216,340**	216,664	**214,181**	215,506
T4-02	**219,954**	221,504	218,414	**216,350**
T4-03	222,980	**218,718**	220,409	**216,814**
T4-04	**219,139**	222,581	**217,756**	218,094
T4-05	**220,460**	221,212	**219,281**	219,683
T4-06	219,149	**219,112**	216,181	**215,830**
T4-07	219,666	**218,058**	**216,285**	216,839
T4-08	**216,103**	218,168	**215,807**	215,936
T4-09	223,116	**220,051**	221,855	**215,975**
T4-10	219,098	**217,057**	216,856	**214,176**
T5-01	347,529	**346,924**	347,356	**346,111**
T5-02	**349,658**	**349,658**	**349,658**	**349,658**
T5-03	349,642	**348,547**	349,359	**347,590**
T5-04	**345,630**	**345,630**	345,630	**345,175**
T5-05	357,457	**350,891**	356,291	**350,891**
T5-06	343,027	**342,714**	343,027	**342,052**
T5-07	358,069	**352,783**	354,852	**351,553**
T5-08	347,051	**345,996**	347,002	**345,996**
T5-09	351,079	**350,766**	351,079	**350,417**
T5-10	**345,039**	**345,039**	345,039	**343,742**
T6-01	2,394,624	**2,384,517**	2,379,655	**2,370,161**
T6-02	2,450,322	**2,430,110**	2,432,481	**2,392,949**
T6-03	2,400,259	**2,390,905**	2,389,128	**2,380,847**
T6-04	2,390,995	**2,390,943**	2,390,995	**2,370,129**
T6-05	2,397,184	**2,384,236**	2,387,226	**2,372,693**
T6-06	2,442,339	**2,426,068**	2,416,792	**2,390,244**
T6-07	**2,393,514**	2,404,469	2,389,038	**2,382,052**
T6-08	2,393,010	**2,389,088**	2,384,886	**2,373,894**
T6-09	2,428,272	**2,422,976**	2,415,699	**2,389,409**
T6-10	2,386,473	**2,382,799**	2,377,996	**2,362,586**

B Tables of Computational Results

Table 3 Costs by type after averaging the scenarios of each instance when the preventive maintenance strategy (PM) is applied.

Instance	Operating costs	Deadhead costs	Maintenance costs	Trip costs	Total costs
T1	11,507	1,342	12,600	247,373	272,822
T2	64,438	25,595	9,800	362,386	462,219
T3	308,384	58,388	29,800	1,025,235	1,421,806
T4	20,137	14,248	12,000	165,630	212,015
T5	41,425	6,459	14,000	284,629	346,513
T6	536,986	63,994	42,200	1,726,235	2,369,415
Σ	982,877	170,025	120,400	3,811,487	5,084,790

Table 4 Costs by type after averaging the scenarios of each instance when the predictive maintenance strategy (PdM) is applied.

Instance	Operating costs	Deadhead costs	Maintenance costs	Trip costs	Total costs
T1	11,507	1,301	11,600	247,373	271,781
T2	64,438	29,939	9,800	362,386	466,564
T3	308,384	51,082	26,600	1,025,235	1,411,300
T4	20,137	15,303	11,000	165,630	212,069
T5	41,425	5,237	13,600	284,629	344,891
T6	536,986	76,852	37,600	1,726,235	2,377,673
Σ	982,877	179,714	110,200	3,811,487	5,084,278

Table 5 Costs by type after averaging the scenarios of each instance when the preventive maintenance strategy with Bayesian inference (PdM-B) is applied.

Instance	Operating costs	Deadhead costs	Maintenance costs	Trip costs	Total costs
T1	11,737	1,227	11,600	247,373	271,937
T2	64,438	21,550	9,800	362,386	458,174
T3	308,384	52,439	26,400	1,025,235	1,412,457
T4	22,151	13,850	10,200	165,630	211,831
T5	41,425	4,523	12,400	284,629	342,976
T6	536,986	74,392	37,400	1,726,235	2,375,013
Σ	985,121	167,981	107,800	3,811,487	5,072,388

Table 6 The cost differences of PdM-B compared to PM and PdM in percent after averaging the scenarios of each instance.

Instance	Cost diff. in %		Cost diff. without fixed costs in %	
	PM	PdM	PM	PdM
T1	−0.32	+0.06	−3.48	+0.64
T2	−0.86	−1.80	−4.05	−8.05
T3	−0.66	+0.08	−2.36	+0.30
T4	−0.09	−0.11	−0.40	−0.51
T5	−1.02	−0.56	−5.72	−3.18
T6	+0.24	−0.11	+0.86	−0.41
Combined	−0.24	−0.23	−0.97	−0.93

The Line-Based Dial-a-Ride Problem

Kendra Reiter[1]

Department of Computer Science, University of Würzburg, Germany

Marie Schmidt

Department of Computer Science, University of Würzburg, Germany

Michael Stiglmayr

Department of Mathematics and Informatics, University of Wuppertal, Germany

Abstract

On-demand ridepooling systems offer flexible services pooling multiple passengers into one vehicle, complementing traditional bus services. We propose a transportation system combining the spatial aspects of a fixed sequence of bus stops with the temporal flexibility of ridepooling. In the *line-based Dial-a-Ride problem (liDARP)*, vehicles adhere to a fixed, ordered sequence of stops in their routes, with the possibility of taking shortcuts and turning if they are empty. We propose three MILP formulations for the liDARP with a multi-objective function balancing environmental aspects with customer satisfaction, comparing them on a real-world bus line. Our experiments show that the formulation based on an Event-Based graph is the fastest, solving instances with up to 50 requests in under one second. Compared to the classical DARP, the liDARP is computationally faster, with minimal increases in total distance driven and average ride times.

2012 ACM Subject Classification Applied computing → Transportation

Keywords and phrases DARP, ridepooling, liDARP, public transport, on-demand

Digital Object Identifier 10.4230/OASIcs.ATMOS.2024.14

Related Version *ArXiv Version*: `https://arxiv.org/abs/2409.08860`

Supplementary Material
Software (Source Code & Data): `https://github.com/ReiterKM/liDARP` [28], archived at `swh:1:dir:08f1ed7ba3a09a49eaffffffae74e8b1e5e7dd74`

1 Introduction

Line-based bus services are able to pool a large number of transportation requests along popular trajectories, thus contributing towards reducing mobility-related emissions when people decide to take the bus instead of the car. However, when there is little demand (e.g., in rural areas or during off-peak periods), buses often run infrequently and almost empty, so that the described benefits do not materialize. Ridepooling approaches, where multiple passengers are pooled into a shared vehicle, accepting a slight detour compared to their direct route, are often proposed to complement line-based bus services for these scenarios. These approaches are inefficient where they do not succeed to pool requests sufficiently.

We formalize a conceptual approach called *line-based ridepooling* which combines the spatial aspect of a classical line-based bus service with the temporal flexibility of on-demand ridepooling, inspired by real-life examples, including the FLEX'HOP 72[2] in France, and the NAHBUS[3] and the Rufbus[4] in Germany. In line-based ridepooling, we consider a fixed and ordered sequence of bus stops (defined, e.g., by a prior operating regular bus line) which we

[1] corresponding author

[2] `https://www.cts-strasbourg.eu/fr/se-deplacer/transport-a-la-demande/`

[3] `https://www.nahbus.de/rufbus`

[4] `https://rufbus.nordfriesland.de/Rufbus-Nördliches-u-südliches-Nordfriesland/`

24th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2024).
Editors: Paul C. Bouman and Spyros C. Kontogiannis; Article No. 14; pp. 14:1–14:20
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

use as pick-up and drop-off locations. In contrast to a classical line-based bus, our vehicles have the flexibility to skip stops and take shortcuts, enabling tailored pick-up and drop-off times dependent on the specific customer requests. However, in contrast to ridepooling, we use the spatial structure of the given line as a sort of service promise, ensuring that passengers are only transported towards their destination along the line. In particular, vehicles may not turn with passengers on board. We call this the *directionality property*. We aim to achieve a transportation mode which is more efficient and provides a higher quality of service than the classical line-based bus, especially in areas with low demand or during off-peak times.

In this paper, we define a new optimization problem to serve passenger requests in line-based ridepooling. Due to its similarity to the general Dial-a-Ride problem (DARP), from which it differs by the directionality property and the underlying geography, we call this problem the *line-based Dial-a-Ride problem (liDARP)*. Here, we study the static variant, where all requests are known ahead-of-time, and consider a homogeneous fleet of vehicles.

We observe that, due to the directionality property, each vehicle route can be decomposed into a number of sublines, separated by vehicle turns, with each transported passenger assigned to exactly one subline. We introduce and compare three mixed-integer linear programming (MILP) formulations for the liDARP that exploit this property. The first formulation, presented in Section 4.1, explicitly models sublines and the assignment of passengers to them. The second and third formulation, presented in Section 4.2 and Section 4.3, are based on Cordeau's classic 3-index Location-Based formulation [7] and the Event-Based model introduced by Gaul et al. in [14]. Section 5 discusses computational results.

Our contribution is threefold: First, we present a general problem definition for the liDARP, an approach to organizing passenger transport with the potential to combine benefits from a line-based public transport and on-demand transportation. Second, we develop and present three MILP formulations for the liDARP. Third, we compare these three models on synthetic test instances.

2 Related Work

Traditional modes for passenger transportation like the bus, metro, or train, operate based on lines (prescribing the sequence of stops visited) and timetables (prescribing the timing of each stop) or frequencies (prescribing the distance to be kept between individual vehicles on a line). While public transport planning often takes a network perspective, there are also many contributions that study timetabling or frequency setting on an individual line with the objective to find an optimal balance between service quality and operator cost, see, e.g., [18, 19] and the references therein. Gkiotsalitis et al. [15] present a model that allows to establish regularly operating sublines within a longer line to deal with inhomogenous demand along the line. Aktaş et al. [1] study a situation where selected stops are assigned to an *express service*, forming a shorter and quicker route. Their goal is to determine which vehicles should perform this express service during morning rush hour, based on expected demand. While still a rather uncommon strategy during the *planning* of public transport operations, short-turning and stop-skipping are common *control* strategies in transit systems to mitigate effects like vehicle bunching and overcrowding, see [18].

The literature on Dial-a-Ride problems (synonymously called *ridepooling*, *on-demand* bus services, or *demand-responsive transport*) is extensive, with in-depth overviews of the current state being provided by Cordeau and Laporte [9] (until 2007) and Ho et al. [17] (2007 until 2018). Typography and variants are discussed in Molenbruch et al. [20], whence this paper is concerned with the static, homogeneous, multi-objective approach, compromising the conflicting goals of system efficiency (including environmental aspects) and user experience.

Early approaches towards exact solution methods to the DARP were carried out by Psaraftis in [25] and [26]. Cordeau [7] proposes a 3-index arc-based mixed-binary linear program for the standard DARP, which was adapted to a 2-index formulation by Røpke et al. [30]. Røpke et al. propose a branch-and-cut approach which is tested on a large number of benchmark instances. Parragh [22] constructs further valid inequalities related to capacity restrictions, integrating these into a branch-and-cut framework as well as a variable neighborhood search heuristic, based on both the 3-index and 2-index formulations. Gschwindt and Irnich [16] develop an exact branch-and-cut-and-price approach, which solves all instances of the benchmark set introduced by [30] exactly. Recently, Rist and Forbes [29] propose a branch-and-cut framework where a DARP route is broken in multiple *fragments*, which are paths between a request's pick-up and drop-off where the vehicle has a non-empty load. Then, a route is created as a combination of fragments. Gaul et al. [14] propose a new MILP formulation, relying on an *Event-Based* graph with nodes representing pick-up/drop-off events denoting a feasible user allocation of the corresponding vehicle and edges connecting feasible transitions between events.

Next to exact methods, many papers consider heuristic solution methods to solve the DARP, including metaheuristics such as simulated annealing [4, 27], adaptive large neighborhood search [23, 24, 31], and tabu search [3, 8].

We are aware of only three publications where the DARP is studied in combination with an underlying line structure: Archetti et al. [2] restate and prove results from the dissertation of Busch [6], showing that the Vehicle Routing Problem on the line is NP-hard, both with an unlimited and a limited fleet of fixed capacity. A complexity classification of DARP variants has been proposed by de Paepe et al. [10], establishing a scheme akin to scheduling problems. They examine variants on the line geography, showing that the DARP on the line with one vehicle of capacity one is solvable in polynomial time. The DARP on a line with multiple homogeneous vehicles of fixed capacity ≥ 1 is NP-complete, which has been shown by Bjelde et al. [5] based on a reduction from the Circular Arc Coloring problem. All three papers are focused on exploring the complexity of the problem, where they consider only the special case with equally spaced stations and do not allow for shortcuts.

3 Problem Description

We consider a set of κ vehicles of capacity $Q_{\max}$ that operate on a bus line, specified by a sequence of bus stops $H = (1, \dots, n)$, to transport m stop-to-stop passenger requests R.

The vehicles do not need to traverse the whole line in each route, but are allowed to take short-cuts (including skipping stops at which no passenger wants to board or alight), to wait, and to turn at any stop, the latter of which may not be done with passengers on board. In this way, we guarantee that the *directionality property* is fulfilled, i.e., each passenger, at all times, is transported towards their direction with respect to the sequence of stops defined by the bus line. Pairwise (time) distances $t_{i,j}$ between all stops $i, j \in H$ are given, with $t_{i,i} := t_{\text{turn}}$ denoting the turn time at $i \in H$. These distances respect the triangle inequality.

Each request $r \in R$ specifies an origin stop $o_r \in H$, a destination stop $d_r \in H$, a time window $[e_r,\ l_r]$, a load (number of passengers in the request) q_r, and a service time b_r for boarding and alighting. We assume that boarding is synchronous, i.e., if one request's destination is another request's origin, and both requests are transported by the same vehicle, we require that the first request alights before the second boards. This reflects the widely accepted standard boarding procedure on public transit systems. Furthermore, passengers do not transfer between vehicles.

We make two *service promises* to our accepted passengers regarding 1) their total travel time and 2) their waiting time that have to be respected. For the former, we guarantee that the passenger's total ride time L_r will not exceed the time needed to travel the direct route (between their origin and destination) by a pre-specified factor, the *excess factor* α, i. e., $L_r^{\max} := \alpha \cdot t_{o_r,d_r}$. For the latter, we ensure that the actual pick-up (resp. drop-off) time is not more than β minutes later (resp. earlier) than the specified earliest pick-up (resp. latest drop-off) time.

Our objective is to create a reliable service for customers and, at the same time, integrate environmental aspects by reducing emissions compared to passengers travelling in their own vehicles. Therefore, our objective function is composed of two weighted components: the number of *accepted* passengers and the *saved distance* (i. e., the difference between the sum of direct distances between all origins and destinations and the total distance driven by our vehicles), which we want to maximize. The optimization problem now consists of deciding which passenger requests are accepted, and which are rejected, to assign accepted requests to one of the κ vehicles, and to plan the routes of these vehicles.

The above-defined problem is a variant of the Dial-a-Ride problem: removing the restriction that vehicles may only turn without passengers on board, it reduces to a (standard) DARP. Given that our vehicle's operations are constrained by the line, we call our problem the line-based Dial-a-Ride problem (liDARP).

4 MILP Formulations for the liDARP

The underlying line structure, which defines an order of bus stops, combined with the directionality property, allows us to divide the route of each vehicle into a number of *sublines*: a sequence of stops at which a vehicle stops to pick-up or drop-off passengers. The first subline is initialized when a vehicle starts its route and a new subline starts after each of the vehicle's turns. We split the set of sublines S into *ascending sublines* (S^{asc}), travelling from a stop i to j with $i < j$, and *descending sublines* (S^{desc}), travelling in the opposite direction.

Similarly, we divide the passenger requests $r \in R$ into *ascending requests* (R^{asc}) and *descending requests* (R^{desc}). As passengers may not be on board when the vehicle turns, each accepted request can be assigned to exactly one subline, with ascending requests assigned to ascending sublines and descending requests assigned to descending sublines.

In Section 4.1, we exploit these properties to propose a *Subline-Based* MILP for the liDARP, explicitly modelling sublines and passenger assignments to sublines. In Section 4.2 and Section 4.3, we show that the sublines can also be used to simplify MILP formulations for the standard DARP.

Note that the sublines in the liDARP are similar to the so-called *fragments* proposed by Rist and Forbes [29] in their branch-and-cut approach for the DARP. Namely, a subline can be further subdivided into fragments, which start with a pick-up node and end when the vehicle is empty.

4.1 Subline-Based Formulation

The *Subline-Based formulation* relies on the concept of a subline. Given the set of vehicles K, we assign each $k \in K$ a set of sublines S and use binary variables $y_i^{s,k}$ to indicate whether subline s of vehicle k stops at bus stop i. The route of every subline s of vehicle k is encoded by the binary variables $x_{i,j}^{s,k}$ that denote the path between bus stops $i, j \in H$. Depending on the direction of the subline s, these only need to be defined for $i \leq j$ or $j \leq i$, respectively. Sublines are computed on a vehicle-basis and symmetry breaking constraints are defined on the vehicle's index to remove alternative solutions with equal objectives.

Flow conservation constraints ensure these routes are consistent in each subline and between consecutive sublines. We track the start and end stop of each subline with binary variables $x_{i,i}^{s,k}$, which correspond to sublines s of vehicle k turning at bus stop i.

Requests are assigned to sublines using binary variables $\text{assign}_r^{s,k}$ to indicate if request r is transported by subline s of vehicle k. Note that these only need to be created for pairs (r, s) with both r and s travelling in the same direction. We ensure each passenger is picked up at most once, with the corresponding subline stopping at both the origin and destination stop. The underlying line structure determines each subline's pick-up and drop-off sequence, allowing capacity constraints to be expressed solely in variables $\text{assign}_r^{s,k}$.

Continuous variables $\text{arr}_i^{s,k}$ and $\text{dep}_i^{s,k}$ model the arrival and departure time of subline s of vehicle k at bus stop i, respectively. We introduce constraints to ensure the stopping time is sufficiently long for all assigned passengers who are boarding or alighting the vehicle at a station to do so. Similarly, we ensure that the time between departure at a bus stop i and arrival at the next bus stop j on the vehicle's route is equal to $t_{i,j}$. The departure and arrival times are further constrained by the fact that, when a request is assigned to a subline, the corresponding departure and arrival times have to respect the request's time window and associated service promises. For this, we track the pick-up and arrival time of every request r.

An overview of parameters and variables, and the full model are given in Appendix B.1.

4.2 Location-Based Formulation

Inspired by mathematical programming formulation for the traveling salesperson and vehicle routing problems, Cordeau [7] models the Dial-a-Ride problem using a graph where nodes represent origin o_r and destination d_r locations of requests r and arcs represent direct connections between locations. In principle, traveling between any pair of locations is possible, though many arcs can be removed in a pre-processing step based on time constraints.

To account for the directionality property, we can modify Cordeau's DARP formulation for use in the liDARP as follows: we treat the requests as (general) DARP input, adding modifications to respect the line precedence and to prevent that vehicles turn with passengers on board. Observing that a vehicle may only turn after drop-off location or before a pick-up location, we introduce additional nodes at these bus stops which are used to start or end a turn, allowing us to model our problem based on fewer arcs than the general DARP. A *start-turn* node denotes that a vehicle is turning at a pick-up node (and then starting a new subline), while an *end-turn* node denotes that a vehicle is turning after a drop-off node (and ending the current subline). Similar to the general DARP, many arcs can be excluded by pre-processing based on time windows and service constraints.

Binary variables $x_{i,j}^k$ model whether vehicle k travels from node i to node j with flow constraints ensuring feasible operations. Additional variables and big-M-constraints are needed to keep track of the time at which locations are visited and of vehicle load, so that the requirements with respect to time windows, service promises, and vehicle capacity are ensured. Technical details and the full model are given in Appendix B.2, where we use strengthening techniques based on [11].

4.3 Event-Based Formulation

Encoding feasible user allocations in vehicles as nodes, and constructing only edges between feasible connections, Gaul et al. [14] propose the *Event-Based* graph as a basis to formulate the general DARP as a MILP. Every node in the Event-Based graph represents a $Q_{\max}$-tuple, where the first entry represents the most recent action: a pick-up r^+ or drop-off r^- of a request r, and the remaining entries denote the other passengers on board. The node **0** is used to denote the depot. Finding feasible vehicle routes for the DARP can then be interpreted as a minimum cost circulation flow problem with the additional constraints that each passenger cannot be picked up more than once and that time windows and service constraints need to be respected.

While the number of events grows exponential with $Q_{\max}$, many nodes can already be excluded during the construction of the Event-Based graph due to incompatibility of time windows and service constraints. The directionality property allows us to further reduce this set. In particular, requests i and j cannot be part of the same event if

- one of them is ascending and the other is descending, or
- both requests are ascending (resp. descending) and the request with a later (resp. earlier) starting station cannot board a vehicle with the other already on board due to time window constraints.

The Event-Based model for the DARP can be directly applied to the liDARP by an adapted construction of the Event-Based graph, distinguishing ascending and descending events and connecting only events that preserve the directionality property, hence we do not re-state the formulation here. Moreover, Gaul et al. [12] proposed further arc eliminations.

5 Computational Experiments

In this section, we present numerical experiments for the liDARP on synthetic benchmark instances. For all experiments, we set the passenger load $q_r = 1$ and service time $b_r = 3\,\text{min}$ for all $r \in R$. The service promise parameters were set to a maximum waiting time of $\beta = 15\,\text{min}$ and a maximum exceedance of direct ride time by $\alpha = 3$. The objective function weights were chosen to be $c_1 = 10$ for the number of accepted passenger and $c_2 = 1$ for the saved distance for the computational results.

The models for the Subline-Based and Location-Based formulation were implemented in Python 3.11 using Gurobi 10.0. The Event-Based formulation was implemented in C++ 17 using CPLEX 22.1, based on the code by Gaul et al. [13]. The computations are carried out using a 12th Gen Intel Core i7-1260P CPU, running at 2.10 GHz with 32 GB RAM. For all runs, we set the solver timeout to 60 min and repeated the calculation five times, averaging the runtimes.

5.1 Benchmark Instances

We create new benchmark instances specifically for the liDARP using the existing bus stops of bus line 6 in Würzburg, Germany, as pictures in Figure 1, with 16 stops connecting the city center to a residential area. We calculate bus stop distances using OpenStreetMap [21], assuming vehicles can take shortcuts and rounding to the nearest minute.

For the given bus stops H, we generate requests that uniformly choose a pick-up and drop-off stop in H. We generate an equal amount of requests per time window type, picking the (earliest) pick-up time or (latest) drop-off time uniformly in the interval $[0, 480]$, corresponding to an operation of 8 hours. The vehicles have a capacity of $Q_{\max} \in \{3, 6\}$ and take $t_{\text{turn}} = 3\,\text{min}$ to turn.

We generated 14 instances on the given sequence H, varying from 16 requests with 2 vehicles to 50 requests with 5 vehicles, following the sizing of the well-known benchmark instances by Cordeau [7] for the classical DARP. The instance names consist of a prefix ‘w’ (for Würzburg), followed by two numbers, where the first indicates the number of vehicles and the second denotes the number of requests.

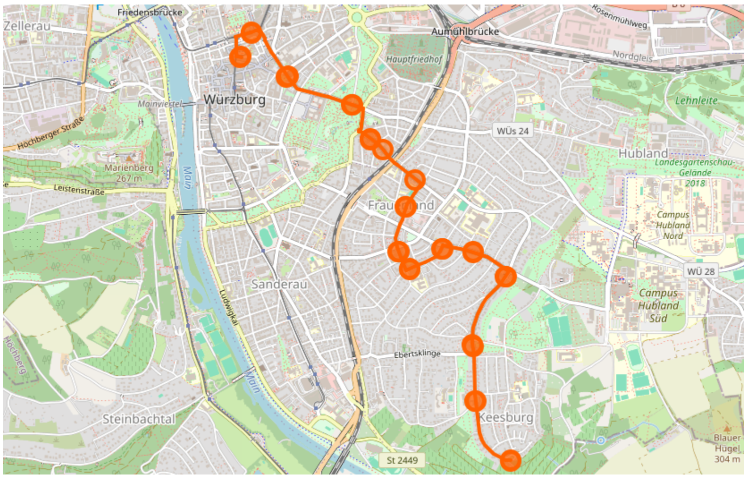


Figure 1 Route of bus line 6 in Würzburg, Germany, from [21].

5.2 Results

In this section, we first compare the three proposed MILP formulations using the benchmark bus line test instances. Second, we assess the trade-off between environmental savings and customer satisfaction. Lastly, we compare the liDARP model to the classical DARP model to evaluate its competitiveness.

Computational Time

Figure 2 shows the computational time per benchmark instance for all three formulations.

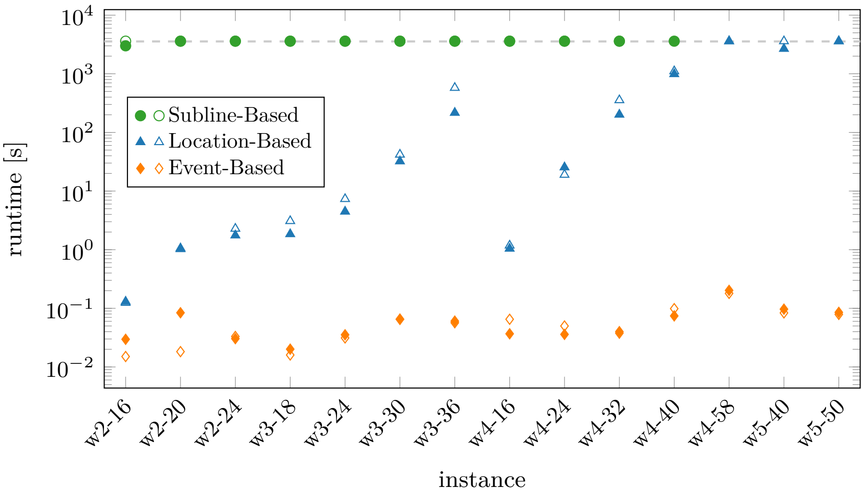


Figure 2 Runtime of the three formulations on the benchmark instances. Solid markers denote $Q_{\max} = 3$, unfilled markers $Q_{\max} = 6$, and the dashed line marks the solver timeout.

The results clearly show that the Event-Based model outperforms the Location-Based and Subline-Based models in all instances, for either capacity. For $Q_{\max} = 3$, the Subline-Based model reached the timeout for all but one instance, w2-16, and we were not able to compute a solution for the three largest instances due a lack of available memory. For $Q_{\max} = 6$, the largest possible instance the model could solve was w4-32.

Observing the Location-Based model, we see a step-structure, where the runtime significantly increases with the number of requests, then decreases as the instances switch to the next-largest number of vehicles. We see a similar effect in the Event-Based model between instances w4-58 and w5-40. This is supported by the model size differences reported in Table 1, which strongly correlate to the number of passenger requests.

In the two instances where the Location-Based model reached timeout for $Q_{\max} = 3$, namely w4-58 and w5-50, we note that although the achieved a relative MIP gap at timeout was greater than 1, the objective value found was within 8 % and even 0 % of the optimum, respectively.

Table 1 Number of constraints and variables for the benchmark test instances with $Q_{\max} = 3$, ordered by the number of requests. SB = Subline-Based, LB = Location-Based, EB = Event-Based.

	Num. Constraints			Num. Boolean Var.			Num. Cont. Var.		
Inst.	SB	LB	EB	SB	LB	EB	SB	LB	EB
w2-16	45 042	553	317	20 030	968	189	1952	150	53
w4-16	90 388	787	335	40 188	1868	196	3872	154	56
w3-18	83 315	752	401	35 841	1731	238	3276	170	64
w2-20	67 290	689	497	28 062	1372	294	2440	186	76
w2-24	96 434	825	746	38 462	2062	430	2928	222	106
w3-24	144 443	998	830	57 117	3168	449	4368	224	119
w4-24	197 732	1171	769	79 228	4116	437	5808	226	109
w3-30	227 963	1244	956	87 213	4473	610	5460	278	122
w4-32	352 756	1555	1174	133 500	6792	706	7744	298	149
w3-36	340 895	1490	1627	127 749	6564	935	6552	332	199
w4-40	577 636	1939	1867	214 204	10 668	1117	9680	370	223
w5-40	722 789	2224	1657	267 755	12 945	1065	12 080	372	188
w5-50	1 227 649	2774	2419	453 155	20 885	1602	15 100	462	249
w4-58	1 400 268	2803	3282	517 020	21 976	2154	14 036	532	326

The Subline-Based model requires a significantly higher number of resources, with the number of constraints and boolean variables exceeding those of both the Location-Based and Event-Based models by factors of 100 and 10, respectively. Notably, the Location-Based model uses 10 times more boolean variables than the Event-Based model, but both require a similar amount of continuous variables.

In general, this result is not surprising, as Gaul et al. [14] demonstrated the computational efficiency of the Event-Based graph in a MILP formulation for the classical DARP, as many complicating constraints are implicitly encoded in the underlying network structure.

All following experiments are carried out with $Q_{\max} = 3$.

Trade-off Analysis

To evaluate the trade-off between environmental savings and customer attractiveness in our chosen objective function, we compare three different settings: in the *environmentally focused* setting, we use objective weights $c_1 = 1$, $c_2 = 10$, placing an emphasis on the distance saved, whilst in the *customer focused* setting, we use weights $c_1 = 10, c_2 = 1$, emphasising the number of transported passengers. We also include a setting with equal weights as a base case. The trade-off between objective function components is visualized in Figure 3.

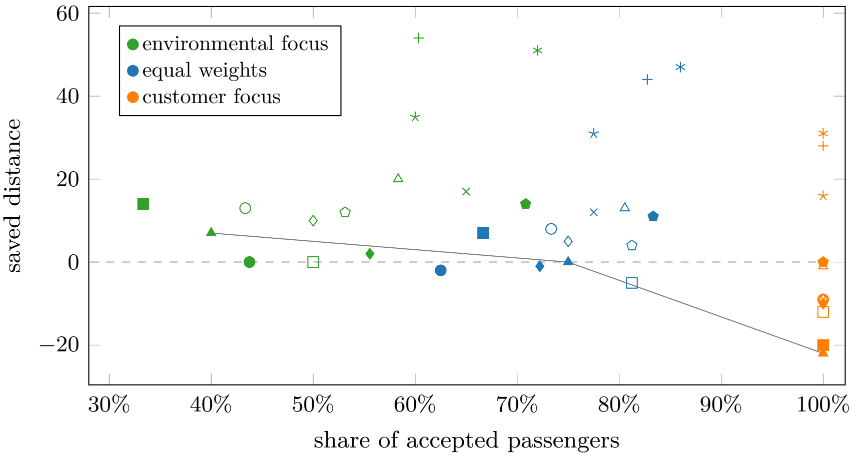


Figure 3 Objective Function parameters for varying weights. Each marker shape represents one benchmark instance.

Note that both objective functions are considered as maximizing objectives. A positive saved distance is preferred, as this corresponds to more direct passenger kilometers saved than total routing costs accumulated. All instances in Figure 3 were solved to optimality.

We observe that, in the customer-focused setting, all passengers are accepted in all instances. In the environmental-focused setting, the saved distance is always non-negative. We have highlighted the approximated Pareto front for a specific instance, w2-20, represented by triangles, by connecting the markers corresponding to the three obtained weighted-sum solutions in Figure 3 to better illustrate the trade-offs. The saved distance decreases from 7 min to −22 min between the environmental-focused and the customer-focused setting, while the share of accepted passengers increases from 40 % to 100 %. The same pattern can be observed for all other benchmark instances. Hence, the proposed objective function is capable of capturing multiple needs and can be adjusted accordingly, dependent on the chosen application.

DARP versus liDARP

Lastly, we compare the liDARP formulation to the classical DARP formulation, where vehicles are allowed to take any route between passenger without needing to adhere to the line structure. We use the above-introduced benchmark instances, extending these to cases with up to 11 vehicles and 132 requests (similar to the extended benchmark set introduced by Røpke et al. [30]). We set $t_{\text{turn}} = 0$ min for both formulations and set the solver timeout to 60 min. The computational time for both models is shown in Figure 4, averaged over five runs. The Event-Based model was used to produce the liDARP results.

We observe that the liDARP model is faster in all instances. While both models' computational time increases with the number of requests, the liDARP was able to solve even the largest instances with over 100 requests in less than 10 s, while the DARP model was aborted at timeout. Examining the objective values, both models accepted all requests in all instances, while they differ in the saved distance, which is visualized in Figure 5. The DARP achieved a higher saved distance in all but one instance, w10-100, which was aborted at timeout, with the average deviation being 3 min and the maximum deviation being a saving of 10 min in instance w11-132. Both models use all available vehicles for all instances.

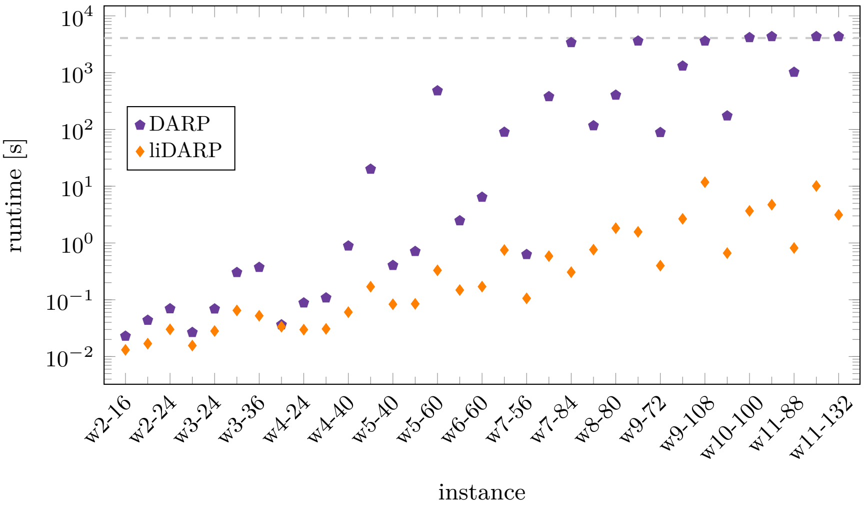


Figure 4 Runtime of the DARP and liDARP on the extended benchmark instances. The dashed line marks the solver timeout.

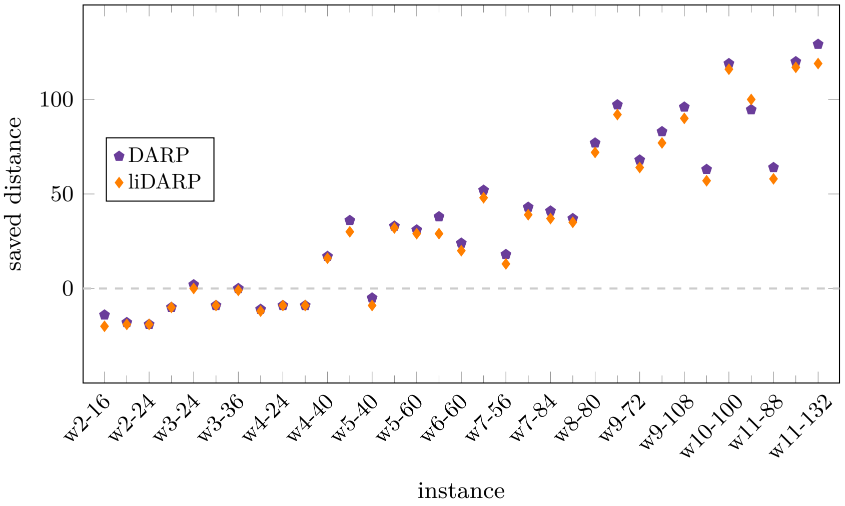


Figure 5 Saved distance of the liDARP and DARP on the extended benchmark instances.

The average ride time, measured as the time between pick-up and drop-off of each request, was marginally higher in the DARP model, with an average increase of 0.14 min. The difference in the average share of empty mileage, defined as the fraction of empty mileage over the total distance, is less than 0.02 on average. Similarly, the difference in average detour, which is the fraction of passenger distance driven to the shortest distances between pick-up and drop-off, is less than 0.025 on average.

Examining each DARP solution, we count the number of requests which are, at least for a portion of their trip, travelling *away* from their destination, i.e., violating the directionality property. The only instances where there are no such violations are w3-18, w3-30, and w4-32. On average, 7.7 % of passengers travel in the opposite direction for at least a portion of their trip, with the largest amount being 18.9 % in instance w9-90. We hypothesize that these routes will likely be viewed as unnecessary by customers, even if they are the most efficient amongst all possible connections.

While there is a difference in saved distance and ride time, passengers do not have to accept significant detours when using the liDARP compared to the DARP model, and the vehicles travel without passengers for a similar amount of time.

6 Conclusion

We present the line-based Dial-a-Ride problem (liDARP), wherein ridepooling vehicles operate on-demand on a sequence of bus stations, adhering to the directionality property, time and capacity constraints, and our service promises. The efficiency of this approach is validated through numerical computations on benchmark instances derived from on a real-life bus line in Würzburg, Germany. Our multi-objective approach successfully balances environmental concerns, by reducing the total distance travelled compared to individual passenger trips, and the attractiveness, measured in the total amount of passengers accepted.

The advantages of the liDARP additionally include the possibility to use existing infrastructure, being based on a sequence of bus stops, which may increase utilization in off-peak times or in areas with low demand. Thus, we provide a system that can serve as an alternative to public transport, providing flexibility to its customers to improve attractiveness.

Our approach can be extended to inhomogeneous vehicles with varying capacities, to determine where each vehicle is best allocated to serve the customer base. Future research could explore varying demand scenarios, such as operating a feeder line to a train station or considering rush-hour induced fluctuations in demand. Finally, this paper focuses on the static variant of the liDARP, where all requests are known in advance. Investigating the liDARP under unknown and dynamic demand may provide insights into its practical applicability and competitiveness in a real-world setting.

References

1 Dilay Aktas, Pieter Vansteenwegen, and Kenneth Sörensen. A demand-responsive bus system for peak hours with capacitated vehicles. In *Proc. 11th Triennial Symposium on Transportation Analysis conference (TRISTAN XI)*, Mauritius Island, 2022. TRISTAN.

2 Claudia Archetti, Dominique Feillet, Michel Gendreau, and M. Grazia Speranza. Complexity of the VRP and SDVRP. *Transportation Research Part C: Emerging Technologies*, 19(5):741–750, August 2011. doi:10.1016/j.trc.2009.12.006.

3 Andrea Attanasio, Jean-François Cordeau, Gianpaolo Ghiani, and Gilbert Laporte. Parallel Tabu search heuristics for the dynamic multi-vehicle dial-a-ride problem. *Parallel Computing*, 30(3):377–387, March 2004. doi:10.1016/j.parco.2003.12.001.

4 John W. Baugh, Gopala Krishna Reddy Kakivaza, and John R. Stone. Intractability of the Dial-a-Ride Problem and a Multiobjective Solution Using Simulated Annealing. *Engineering Optimization*, 30(2):91–123, February 1998. doi:10.1080/03052159808941240.

5 Antje Bjelde, Jan Hackfeld, Yann Disser, Christoph Hansknecht, Maarten Lipmann, Julie Meißner, Miriam Schlöter, Kevin Schewior, and Leen Stougie. Tight bounds for online tsp on the line. *ACM Transactions on Algorithms (TALG)*, 17(1):1–58, 2021. doi:10.1145/3422362.

6 Ingrid Busch. *Vehicle routing on acyclic networks.* Dissertation, The Johns Hopkins University, Baltimore, Maryland, 1991.

7 Jean-François Cordeau. A Branch-and-Cut Algorithm for the Dial-a-Ride Problem. *Operations Research*, 54(3):573–586, 2006. doi:10.1287/opre.1060.0283.

8 Jean-François Cordeau and Gilbert Laporte. A tabu search heuristic for the static multi-vehicle dial-a-ride problem. *Transportation Research Part B: Methodological*, 37(6):579–594, 2003. doi:10.1016/S0191-2615(02)00045-0.

9 Jean-François Cordeau and Gilbert Laporte. The dial-a-ride problem: models and algorithms. *Annals of Operations Research*, 153(1):29–46, 2007. doi:10.1007/s10479-007-0170-8.

10 Willem E. de Paepe, Jan Karel Lenstra, Jiri Sgall, René A. Sitters, and Leen Stougie. Computer-aided complexity classification of dial-a-ride problems. *INFORMS Journal on Computing*, 16(2):120–132, 2004. doi:10.1287/ijoc.1030.0052.

11 Martin Desrochers and Gilbert Laporte. Improvements and extensions to the Miller-Tucker-Zemlin subtour elimination constraints. *Operations Research Letters*, 10(1):27–36, 1991. doi:10.1016/0167-6377(91)90083-2.

12 Daniela Gaul, Kathrin Klamroth, Christian Pfeiffer, Arne Schulz, and Michael Stiglmayr. A Tight Formulation for the Dial-a-Ride Problem, 2023. arXiv:2308.11285.

13 Daniela Gaul, Kathrin Klamroth, and Michael Stiglmayr. Solving the Dynamic Dial-a-Ride Problem Using a Rolling-Horizon Event-Based Graph. In *21st Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2021)*, volume 96 of *Open Access Series in Informatics (OASIcs)*, pages 8:1–8:16, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/OASIcs.ATMOS.2021.8.

14 Daniela Gaul, Kathrin Klamroth, and Michael Stiglmayr. Event-based MILP models for ridepooling applications. *European Journal of Operational Research*, 301(3):1048–1063, 2022. doi:10.1016/j.ejor.2021.11.053.

15 Konstantinos Gkiotsalitis, Marie Schmidt, and Evelien van der Hurk. Subline frequency setting for autonomous minibusses under demand uncertainty. *Transportation Research Part C: Emerging Technologies*, 135:103492, 2022. doi:10.1016/j.trc.2021.103492.

16 Timo Gschwind and Stefan Irnich. Effective handling of dynamic time windows and its application to solving the dial-a-ride problem. *Transportation Science*, 49(2):335–354, 2015. doi:10.1287/trsc.2014.0531.

17 Sin C. Ho, Wai Yuen Szeto, Yong-Hong Kuo, Janny M.Y. Leung, Matthew Petering, and Terence W.H. Tou. A survey of dial-a-ride problems: Literature review and recent developments. *Transportation Research Part B: Methodological*, 111:395–421, 2018. doi:10.1016/j.trb.2018.02.001.

18 Omar J. Ibarra-Rojas, Felipe Delgado, Ricardo Giesen, and Juan Carlos Muñoz. Planning, operation, and control of bus transport systems: A literature review. *Transportation Research Part B: Methodological*, 77:38–75, 2015. doi:10.1016/j.trb.2015.03.002.

19 Pei Liu, Marie Schmidt, Qingxia Kong, Joris Camiel Wagenaar, Lixing Yang, Ziyou Gao, and Housheng Zhou. A robust and energy-efficient train timetable for the subway system. *Transportation Research Part C: Emerging Technologies*, 121:102822, 2020. doi:10.1016/j.trc.2020.102822.

20 Yves Molenbruch, Kris Braekers, and An Caris. Typology and literature review for dial-a-ride problems. *Annals of Operations Research*, 259(1):295–325, 2017. doi:10.1007/s10479-017-2525-0.

21 OpenStreetMap contributors. Planet dump retrieved from https://planet.osm.org . https://www.openstreetmap.org, 2017.

22 Sophie N. Parragh. Introducing heterogeneous users and vehicles into models and algorithms for the dial-a-ride problem. *Transportation Research Part C: Emerging Technologies*, 19(5):912–930, 2011. doi:10.1016/j.trc.2010.06.002.

23 Sophie N. Parragh and Verena Schmid. Hybrid column generation and large neighborhood search for the dial-a-ride problem. *Computers & Operations Research*, 40(1):490–497, 2013. doi:10.1016/j.cor.2012.08.004.

24 Christian Pfeiffer and Arne Schulz. An ALNS algorithm for the static dial-a-ride problem with ride and waiting time minimization. *OR Spectrum*, 44(1):87–119, 2022. doi:10.1007/s00291-021-00656-7.

25 Harilaos N. Psaraftis. A Dynamic Programming Solution to the Single Vehicle Many-to-Many Immediate Request Dial-a-Ride Problem. *Transportation Science*, 14(2):130–154, 1980. doi:10.1287/trsc.14.2.130.

26 Harilaos N. Psaraftis. An Exact Algorithm for the Single Vehicle Many-to-Many Dial-A-Ride Problem with Time Windows. *Transportation Science*, 17(3):351–357, 1983. doi:10.1287/trsc.17.3.351.

27 Line Blander Reinhardt, Tommy Clausen, and David Pisinger. Synchronized dial-a-ride transportation of disabled passengers at airports. *European Journal of Operational Research*, 225(1):106–117, 2013. doi:10.1016/j.ejor.2012.09.008.

28 Kendra Reiter. The line-based Dial-a-Ride problem. Software, swhId: swh:1:dir:08f1ed7ba3a09a49eafffffae74e8b1e5e7dd74 (visited on 2024-08-20). URL: https://github.com/ReiterKM/liDARP.

29 Yannik Rist and Michael A. Forbes. A New Formulation for the Dial-a-Ride Problem. *Transportation Science*, 55(5):1113–1135, 2021. doi:10.1287/trsc.2021.1044.

30 Stefan Ropke, Jean-François Cordeau, and Gilbert Laporte. Models and branch-and-cut algorithms for pickup and delivery problems with time windows. *Networks*, 49(4):258–272, 2007. doi:10.1002/net.20177.

31 Stefan Ropke and David Pisinger. An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows. *Transportation Science*, 40(4):455–472, 2006. doi:10.1287/trsc.1050.0135.

A Variable Overview

Table 2 Summary of liDARP parameters.

Notation	Definition
H	set of bus stops, $\{1, \dots, n\}$
K	set of vehicles, $\{1, \dots, \kappa\}$
R	set of passenger requests, $\{1, \dots, m\}$
R^{asc}	set of passenger requests travelling in ascending direction
R^{desc}	set of passenger requests travelling in descending direction
o_r	origin stop of passenger request r
d_r	destination stop of passenger request r
q_r	load of request r
e_r	earliest departure time of request r
l_r	latest arrival time of request r
b_r	service time for request r
$t_{i,j}$	travel time from bus stop i to j
α	service promise constant relating to maximum ride time, excess factor
β	service promise constant relating to maximum wait time
L_r	total ride time of request r
$L_r^{\max}$	maximum ride time of request r, dependent on α and β
t_{turn}	time it takes for a vehicle to turn around
c_1, c_2	objective weights

B MILP Formulations

B.1 Subline-Based Formulation

In this section, we present the MILP model for the Subline-Based formulation introduced in Section 4.1. All parameters and variables are summarized in Table 3.

Table 3 Summary of notation for the Subline-Based model.

Notation	Definition
Parameters	
S	set of sublines, $\{1, \dots, \sigma\}$
S^{asc}	set of sublines travelling in ascending direction
S^{desc}	set of sublines travelling in descending direction
Binary Decision Variables	
$\text{assign}_r^{s,k}$	1 if request r is assigned to subline s of vehicle k
$y_i^{s,k}$	1 if subline s of vehicle k visits node i
start_i^k	1 if node i is the start node of vehicle k
end_i^k	1 if node i is the end node of vehicle k
$x_{i,j}^{s,k}$	1 if node j is visited immediately after node i on subline s of vehicle k
$x_{i,i}^{s,k}$	1 if vehicle k turns at node i after executing subline s
w_{r_1,r_2}	1 if requests r_1 and r_2 are on the same subline of the same vehicle
z_k	1 if vehicle k is in use
Continuous Decision Variables	
$\text{dep}_i^{s,k}$	departure time of subline s of vehicle k at node i
$\text{arr}_i^{s,k}$	arrival time of subline s of vehicle k at node i
pickup_r	pick-up time of request r
arrtime_r	drop-off time of request r

In this model, we explicitly model the path of every subline of each vehicle, using binary variables $y_i^{s,k}$ to denote if subline s of vehicle k stops at bus station i and binary variables $x_{i,j}^{s,k}$ to denote if the direct path from station i to station j is used. Further binary variables start_i^k and end_i^k denote if vehicle k starts and ends at station i, respectively. Then, by tracking the turning stations of every subline s, i.e., stations i where $x_{i,i}^{s,k} = 1$, we track the start and end station of every subline. For every variable which references both passengers and sublines, such as $\text{assign}_r^{s,k}$, we only create those variables where the passenger and subline are travelling in the same direction.

To ensure our model respects the boarding precedence (requests which are alighting leave the vehicle before those boarding can enter), we construct the following three subsets for every request $i \in R$:

- $\text{Prec}_i^{o,o}$, containing all requests j which have the same origin as i, are travelling in the same direction, and should board before i,
- $\text{Prec}_i^{o,d}$, containing all requests j whose destination is at i's origin, are travelling in the same direction, and should alight before i boards, and
- $\text{Prec}_i^{d,d}$, containing all requests j which have the same destination as i, are travelling in the same direction, and should alight before i.

Note that the set $\text{Prec}_i^{d,o}$, which denotes all origin requests that need to be served before request i is dropped-off, is not created as it is always empty due to the boarding assumption.

Furthermore, we track which passengers are pooled together on the same subline of the same vehicle with binary variables w_{r_1,r_2}. These are required, together with the precedence sets, to ensure we allow for sufficient boarding and alighting times per passenger at every stop. To model these, we use big-M constraints with $M_1 := \max_{r \in R} L_r^{\max}$.

Binary variables z_k denote that vehicle k is used by our solution. Here, we use big-M constraints with $M_2 := \max_{r \in R} l_r + (2 \cdot \sigma - 1) \cdot b_r$, where σ denotes the number of sublines, to ensure the arrival and departure time variables for each vehicle are only set if they are also used.

To strengthen the model, we enforce that a vehicle's end stop is placed after it has turned twice, i.e., after two consecutive sublines start and end at the same stop. Then, every following subline is empty and turns at the same stop. This reduces the number of possible solutions with the same objective value.

We define $T^+ := \max_{r \in R} L_r^{\max} + (\sigma - 1)t_{\text{turn}}$ to be the end of service, i.e., when all vehicles end their operation at the latest.

We note that the model's size is dependent on the choice of σ, the number of sublines, which is hard to choose. We set $\sigma = 2 \cdot m$ for all experiments presented here.

The full Subline-Based model is given by:

$$\max_x \; c_1 \cdot \left(\sum_{s \in S} \sum_{k \in K} \sum_{r \in R} \text{assign}_r^{s,k} \cdot t_{o_r, d_r} - \sum_{k \in K} \sum_{s \in S} \sum_{\substack{(i,j) \in H \times H: \\ i \neq j}} x_{i,j}^{s,k} \cdot t_{i,j} \right) + c_2 \cdot \sum_{s \in S} \sum_{k \in K} \sum_{r \in R} \text{assign}_r^{s,k}$$

$$\text{s.t.} \quad \sum_{i \in H} \text{start}_i^k \leq 1 \qquad \forall\, k \in K \tag{1a}$$

$$\sum_{i \in H} \text{end}_i^k = \sum_{i \in H} \text{start}_i^k \qquad \forall\, k \in K \tag{1b}$$

$$\sum_{i \in H} x_{ii}^{s,k} = \sum_{i \in H} \text{start}_i^k \qquad \forall\, k \in K, s \in S \setminus \{\sigma\} \tag{1c}$$

$$x_{i,i}^{\sigma,k} = 0 \qquad \forall\, i \in H, k \in K \tag{1d}$$

$$y_i^{1,k} = \text{start}_i^k + \sum_{j<i} x_{j,i}^{1,k} \qquad \forall\, i \in H, k \in K \tag{1e}$$

$$y_i^{s,k} = x_{i,i}^{s-1,k} + \sum_{j<i} x_{j,i}^{s,k} \qquad \forall\, i \in H, k \in K, s \in S^{\text{asc}} \setminus \{1\} \tag{1f}$$

$$y_i^{s,k} = x_{i,i}^{s-1,k} + \sum_{j>i} x_{j,i}^{s,k} \qquad \forall\, i \in H, k \in K, s \in S^{\text{desc}} \tag{1g}$$

$$y_i^{s,k} = x_{i,i}^{s,k} + \sum_{j>i} x_{i,j}^{s,k} \qquad \forall\, i \in H, k \in K, s \in S^{\text{asc}} \tag{1h}$$

$$y_i^{s,k} = x_{i,i}^{s,k} + \sum_{j<i} x_{i,j}^{s,k} \qquad \forall\, i \in H, k \in K, s \in S^{\text{desc}} \setminus \{\sigma\} \tag{1i}$$

$$y_i^{\sigma,k} = \text{end}_i^k + \sum_{j<i} x_{i,j}^{\sigma,k} \qquad \forall\, i \in H, k \in K \tag{1j}$$

$$z_k = \sum_{i \in H} \text{start}_i^k \qquad \forall\, k \in K \tag{1k}$$

$$\sum_{r: o_r \leq i, d_r > i} \text{assign}_r^{s,k} \leq Q_{\max} \qquad \forall\, k \in K, s \in S^{\text{asc}}, i = 1, \ldots, n-1 \tag{1l}$$

$$\sum_{r: o_r \geq i, d_r < i} \text{assign}_r^{s,k} \leq Q_{\max} \qquad \forall\, k \in K, s \in S^{\text{desc}}, i = 2, \ldots, n \tag{1m}$$

$$z_k \geq y_i^{s,k} \qquad \forall\, i \in H, s \in S, k \in K \tag{1n}$$

$$z_k \geq x_{i,j}^{s,k} \qquad \forall (i,j) \in E, s \in S, k \in K \tag{1o}$$

$$z_k \geq \text{assign}_r^{s,k} \qquad \forall\, r \in R, s \in S, k \in K \tag{1p}$$

$$M_2 \cdot z_k \geq \text{arr}_i^{s,k} \qquad \forall\, i \in H, s \in S, k \in K \tag{1q}$$

$$M_2 \cdot z_k \geq \text{dep}_i^{s,k} \qquad \forall\, i \in H, s \in S, k \in K \tag{1r}$$

$$\text{start}_i^k \geq \sum_{j \in H: j<i} \text{start}_j^{k+1} \qquad \forall\, i \in H, k \in \{1, \dots, \kappa - 1\} \tag{1s}$$

$$\text{dep}_i^{s,k} \geq \text{arr}_i^{s,k} \qquad \forall\, i \in H, s \in S, k \in K \tag{1t}$$

$$\text{arr}_j^{s,k} \geq \text{dep}_i^{s,k} + t_{i,j} \cdot x_{i,j}^{s,k} \qquad \forall (i,j) \in H \times H \text{ with } i < j, s \in S^{\text{asc}}, k \in K \tag{1u}$$

$$\text{arr}_j^{s,k} \geq \text{dep}_i^{s,k} + t_{i,j} \cdot x_{i,j}^{s,k} \qquad \forall (i,j) \in H \times H \text{ with } i > j, s \in S^{\text{desc}}, k \in K \tag{1v}$$

$$\text{arr}_i^{s,k} \geq \text{dep}_i^{s-1,k} + t_{\text{turn}} \cdot x_{i,i}^{s-1,k} \qquad \forall\, i \in H, s \in S^{\text{asc}} \setminus \{1\}, k \in K \tag{1w}$$

$$\text{arr}_i^{s,k} \geq \text{dep}_i^{s-1,k} + t_{\text{turn}} \cdot x_{i,i}^{s-1,k} \qquad \forall\, i \in H, s \in S^{\text{desc}}, k \in K \tag{1x}$$

$$\text{arr}_i^{s,k} \geq 0 \qquad \forall\, i \in H, s \in S, k \in K \tag{1y}$$

$$\sum_{k \in K} \sum_{s \in S} \text{assign}_r^{s,k} \leq 1 \qquad \forall\, r \in R \tag{1z}$$

$$2 \cdot \text{assign}_r^{s,k} \leq y_{o_r}^{s,k} + y_{d_r}^{s,k} \qquad \forall\, k \in K, s \in S, r \in R \tag{1aa}$$

$$\text{assign}_r^{s,k} \cdot (e_r + b_{o_r}) \leq \text{dep}_{o_r}^{s,k} \qquad \forall\, k \in K, s \in S, r \in R \tag{1ab}$$

$$l_r + T^+ \cdot (1 - \text{assign}_r^{s,k}) \geq \text{arr}_{d_r}^{s,k} \qquad \forall\, r \in R, s \in S, k \in K \tag{1ac}$$

$$w_{r_i,r_j} \leq \text{assign}_{r_i}^{s,k} \qquad \forall\, s \in S, k \in K, r_i, r_j \in R \tag{1ad}$$

$$\text{assign}_{r_i}^{s,k} + \text{assign}_{r_j}^{s,k} - 1 \leq w_{r_i,r_j} \qquad \forall\, s \in S, k \in K, r_i, r_j \in R \tag{1ae}$$

$$\text{pickup}_r \geq \sum_{k \in K} \sum_{s \in S} \text{assign}_r^{s,k} \cdot \text{arr}_{o_r}^{s,k} \qquad \forall\, k \in K, s \in S, r \in R \tag{1af}$$

$$\text{arrtime}_r \geq \sum_{k \in K} \sum_{s \in S} \text{assign}_r^{s,k} \cdot \text{arr}_{d_r}^{s,k} \qquad \forall\, k \in K, s \in S, r \in R \tag{1ag}$$

$$b_{r_i} + \text{pickup}_{r_i} - \text{pickup}_{r_j} \leq M_1 \cdot \left(1 - \sum_{k \in K} \sum_{s \in S} w_{r_i,r_j}^{s,k}\right) \forall r_i \in R, r_j \in \text{Prec}_{r_i}^{o,o} \tag{1ah}$$

$$b_{r_i} + \text{arrtime}_{r_i} - \text{pickup}_{r_j} \leq M_1 \cdot \left(1 - \sum_{k \in K} \sum_{s \in S} w_{r_i,r_j}^{s,k}\right) \forall r_i \in R, r_j \in \text{Prec}_{r_i}^{o,d} \tag{1ai}$$

$$b_{r_i} + \text{arrtime}_{r_i} - \text{arrtime}_{r_j} \leq M_1 \cdot \left(1 - \sum_{k \in K} \sum_{s \in S} w_{r_i,r_j}^{s,k}\right) \forall r_i \in R, r_j \in \text{Prec}_{r_i}^{d,d} \tag{1aj}$$

$$\text{dep}_{o_r}^{s,k} \geq \text{assign}_r^{s,k} \cdot (\text{pickup}_r + b_r) \qquad \forall\, r \in R \tag{1ak}$$

$$\text{dep}_{d_r}^{s,k} \geq \text{assign}_r^{s,k} \cdot (\text{arrtime}_r + b_r) \qquad \forall\, r \in R \tag{1al}$$

$$e_{o_r} \leq \text{pickup}_r \leq l_{o_r} \qquad \forall\, r \in R \tag{1am}$$

$$e_{d_r} \leq \text{arrtime}_r \leq l_{d_r} \qquad \forall\, r \in R \tag{1an}$$

$$\text{arrtime}_r + b_r - \text{pickup}_r \leq \alpha \cdot t_{o_r,d_r} \qquad \forall\, r \in R \tag{1ao}$$

Constraint (1a) ensures each vehicle is used at most once and constraint (1b) ensures that a started vehicle also ends at a bus stop. Constraint (1c) makes sure every started vehicle turns around at some station, while (1d) denotes that the last subline of each vehicle does not turn. Constraints (1e) to (1i) are for flow conservation between sublines and their turn stops. The last subline ends at its end stop, which is controlled by (1j). Constraint (1k) tracks the amount of vehicles which are used. The upper capacity of every vehicle is ensured by (1l) and (1m), for both subline directions. Constraints (1n) to (1r) ensure that only vehicles which are started can travel to and between stops. Constraint (1s) is a symmetry breaking constraint which says that vehicles with a smaller index start at a smaller station.

Constraint (1t) ensures a vehicle departs from a bus stop only after it has arrived, while constraints (1u) and (1v) ensure the arrival time at the next bus stop respects the minimum travel time. Constraints (1w) and (1x) take into account the turning time, linking a subline's end time with the subsequent subline's start time at the same stop. Finally, constraints (1y) ensures all times are positive.

Constraint (1z) ensures a passenger is picked up at most once and, by constraint (1aa), only if the subline they are assigned to also stops at their origin and destination. The link between earliest pick-up and departure from the origin times, as well as latest drop-off and arrival at the destination times, is handled with constraints (1ab) and (1ac), respectively. Finally, constraints (1ad) and (1ae) link the variable w_{r_i,r_j} to denote if two passengers are assigned to the same subline of the same vehicle.

Constraints (1af) and (1ag) place a lower bound on the pick-up time and drop-off time of each passenger, dependent on the vehicle's arrival time at the corresponding station. Constraints (1ah) to (1aj) ensure the precedence rules for boarding are respected and add sufficient service times between serving customers. Then, constraints (1ak) and (1al) ensures the vehicle can only depart after the last passenger has fully boarded or alighted. The time windows of each passenger is guaranteed by (1am) and (1an), while the maximum travel time is limited by constraint (1ao).

B.2 Location-Based Formulation

In this section, we describe the construction of the underlying graph for the Location-Based formulation in more detail, as well as presenting the full MILP model. All notation is summarized in Table 4.

We introduce *ascending* bus stops $H^{\text{asc}} := \{h_1, \ldots, h_n\}$ and *descending* bus stops $H^{\text{desc}} := \{h_{n+1}, \ldots, h_{2n}\}$. Bus stops $h_i \in H^{\text{desc}}$ and $h_{n+i} \in H^{\text{asc}}$ are virtual copies of stop $i \in H$. In reality, these may be the same stop on opposite sides of the road, for vehicles travelling in opposite directions. We connect all bus stops $h_i \in H^{\text{desc}}$ with their corresponding $h_{n+i} \in H^{\text{asc}}$, in both directions (corresponding to a turn at station i), and enforce that two stops in either set can only be served in upstream order with respect to the line.

Similar to the classical DARP formulation, for each request $r \in R$, we construct four nodes $o_r, d_r, \bar{o}_r, \bar{d}_r$ in a liDARP-Graph $G_R = (H_R, E_R)$. Here, the nodes o_r and d_r correspond to the classic pick-up and drop-off nodes of r. The node $\bar{o}_r$ is a *start-turn* node, denoting that the vehicle is at o_r, facing the opposite direction, i.e., before it turns and picks up request r at o_r. Similarly, the node $\bar{d}_r$ is an *end-turn* node, denoting that the vehicle is at d_r, has dropped off the request r, and is now turning to continue in the opposite direction. Then,

- if $r \in R^{\text{asc}}$: we construct o_r at the origin $h_{o_r} \in H^{\text{asc}}$, d_r at the destination $h_{d_r} \in H^{\text{asc}}$ stop, $\bar{o}_r$ at $h_{n+o_r} \in H^{\text{desc}}$, and $\bar{d}_r$ at $h_{n+d_r} \in H^{\text{desc}}$.
- if $r \in R^{\text{desc}}$: we construct o_r at the origin $h_{o_r} \in H^{\text{desc}}$, d_r at the destination $h_{d_r} \in H^{\text{desc}}$ stop, $\bar{o}_r$ at $h_{o_r-n} \in H^{\text{asc}}$, and $\bar{d}_i$ at $h_{d_r-n} \in H^{\text{asc}}$.

We set $P := \{o_r : r \in R\}$, $D := \{d_r : r \in R\}$, $\bar{P} := \{\bar{o}_r : r \in R\}$, and $\bar{D} := \{\bar{d}_r : r \in R\}$ to denote the sets of these bus stops. We additionally define time windows on the nodes $\bar{o}_r$ and $\bar{d}_r$ of every request $r \in R$, dependent on the time windows on o_r and d_r, respectively, accounting for the boarding and turn times. We introduce two depots, the *start depot* δ_{start}

Table 4 Summary of notation for the Location-Based model.

Notation	Definition
Parameters	
δ_{start}	start depot
δ_{end}	end depot
P	set of pick-up nodes
D	set of delivery node
$\bar{P}$	set of start-turn nodes, before a pick-up node
$\bar{D}$	set of end-turn nodes, after a drop-off node
N_R	set of all pick-up and delivery nodes in all directions, depending on requests R
H_R	set of all pick-up, delivery, and depot nodes, depending on requests R
E_R	set of all edges between nodes in E_R
Binary Decision Variables	
$x_{i,j}^k$	1 if vehicle k travels on arc $(i,j) \in E_R$
z_k	1 if vehicle k is used
Continuous Decision Variables	
B_i	start of service time at bus stop i
Q_i	passenger load departing bus stop i
L_r	ride time of passenger r

and the *end depot* δ_{end}, where vehicles start and end their route. Let $N_R := P \cup D \cup \bar{P} \cup \bar{D}$ denote all pick-up, drop-off, and turn stops, and $H_R := \{\delta_{\text{start}}, \delta_{\text{end}}\} \cup N_R$ all nodes including the depots.

The edge set $E_R := \bigcup_{i=1}^{10} E_R^i$ between nodes in H_R is constructed as follows:

- $E_R^1 := \{(o_r, d_r) \in P \times D : r \in R\}$, connecting each request's origin with its destination,
- $E_R^2 := \{(v_i, w_j) \in (P \cup D \cup \bar{D}) \times (P \cup D \cup \bar{P}) : h_{v_i}, h_{w_j} \in H^{\text{asc}}, i \neq j, v_i \text{ precedes } w_j, i, j \in R\}$, connecting all *ascending* stops to subsequent stops in the same direction,
- $E_R^3 := \{(v_i, w_j) \in (P \cup D \cup \bar{D}) \times (P \cup D \cup \bar{P}) : h_{v_i}, h_{w_j} \in H^{\text{desc}}, i \neq j, v_i \text{ precedes } w_j, i, j \in R\}$, connecting all *descending* stops to subsequent stops in the same direction,
- $E_R^4 := \{(\bar{o}_r, o_r) \in \bar{P} \times P : r \in R\}$, connecting the start-turn stop of each request with its corresponding origin stop in the opposite direction,
- $E_R^5 := \{(d_r, \bar{d}_r) \in D \times \bar{D} : r \in R\}$, connecting the destination stop of each request with its corresponding end-turn stop in the opposite direction,
- $E_R^6 := \{(v_i, w_j) \in (P \cup D \cup \bar{D}) \times (P \cup D \cup \bar{P}) : v_i, w_j \in H^{\text{asc}}, h_{v_i} = h_{w_j}, i \neq j, \neg(v_i \in D \wedge w_j \in P), e_{v_i} \leq e_{w_j}, l_{v_i} \leq l_{w_j}, i, j \in R\}$, connecting *ascending* stops at the same original physical bus stop if they are compatible regarding their time windows,
- $E_R^7 := \{(v_i, w_j) \in (P \cup D \cup \bar{D}) \times (P \cup D \cup \bar{P}) : v_i, w_j \in H^{\text{desc}}, h_{v_i} = h_{w_j}, i \neq j, \neg(v_i \in P \wedge w_j \in D), e_{v_i} \leq e_{w_j}, l_{v_i} \leq l_{w_j}, i, j \in R\}$, connecting *descending* stops at the same original physical bus stop if they are compatible regarding their time windows,
- $E_R^8 := \{(\delta_{\text{start}}, \delta_{\text{end}})\}$, connecting the starting depot to the ending depot to allow for unused vehicles,
- $E_R^9 := \{(\delta_{\text{start}}, o_r) \in \{\delta_{\text{start}}\} \times P : r \in R\} \cup \{(\delta_{\text{start}}, \bar{o}_r) \in \{\delta_{\text{start}}\} \times \bar{P} : r \in R\}$, connecting the start depot to all pick-up locations and their start-turn stops,
- $E_R^{10} := \{(d_r, \delta_{\text{end}}) \in D \times \{\delta_{\text{end}}\} : r \in R\} \cup \{(\bar{d}_r, \delta_{\text{end}}) \in \bar{D} \times \{\delta_{\text{end}}\} : r \in R\}$, connecting all drop-off locations and their end-turn stops to the end depot.

Here, we write v *precedes* w to denote that the bus stations corresponding to v precedes that corresponding to w with respect to the corresponding line direction. We use $E_{\text{turn}} := E_R^4 \cup E_R^5$ to denote all the edges on which the vehicles turn. Each edge is only added once, even if it appears in multiple sets. The travel time of the edges is given by the original network, where a turn takes t_{turn} and travel between two pick-ups or drop-offs at the same physical stop is instantaneous.

In our model, the binary variable $x_{i,j}^k$ denotes if a vehicle k travels on defined arcs $(i,j) \in E_R$. Variables z_k denote if vehicle k is used in the solution.

We define that the vehicle loads $q_{\delta_{\text{start}}} := q_{\delta_{\text{end}}} = 0$, $q_i := 1$ for all $i \in P$ and $q_i := -1$ for all $i \in D$. Additionally, we set the service times $b_{\delta_{\text{start}}} := b_{\delta_{\text{end}}} = 0$ and $b_i = 0$ for all $i \in \bar{P} \cup \bar{D}$. Let Q_i denote the passenger load of a vehicle departing a stop i and let the continuous variable B_i denote the start of service time at stop i. Note that these do not require an index for the vehicle k as each node can be visited by at most one vehicle and the vehicles are homogeneous with a maximum capacity $Q_{\max}$, as has been discussed in [7]. We require that $Q_{\delta_{\text{start}}}^k := Q_{\delta_{\text{end}}}^k = 0$ for all $k \in K$, thus removing these variables from the model.

To strengthen the model, we introduce a symmetry breaking constraint which enforces that vehicles of lower index are used first. Then, the full Location-Based model is given by:

$$\max_x \quad c_1\left(\sum_{k\in K}\sum_{i\in P} t_{i,i+m}\cdot x_{i,i+m}^k - \sum_{k\in K}\sum_{\substack{j\in H_R:\\(i,j)\in E_R}}\sum_{i\in H_R} t_{i,j}\cdot x_{i,j}^k\right) + c_2\sum_{k\in K}\sum_{\substack{j\in H_R:\\(i,j)\in E_R}}\sum_{i\in P} x_{i,j}^k$$

$$\text{s.t.} \quad \sum_{k\in K}\sum_{\substack{j\in H_R:\\(i,j)\in E_R}} x_{i,j}^k \le 1 \qquad \forall\, i \in P \tag{2a}$$

$$\sum_{\substack{j\in H_R:\\(i,j)\in E_R}} x_{i,j}^k - \sum_{\substack{j\in H_R:\\(m+i,j)\in E_R}} x_{m+i,j}^k = 0 \qquad \forall\, i\in P, k\in K \tag{2b}$$

$$\sum_{j\in P\cup\bar{P}} x_{\delta_{\text{start}},j}^k = 1 \qquad \forall\, k\in K \tag{2c}$$

$$\sum_{i\in D\cup\bar{D}} x_{i,\delta_{\text{end}}}^k = 1 \qquad \forall\, k\in K \tag{2d}$$

$$\sum_{\substack{j\in H_R:\\(j,i)\in E_R}} x_{j,i}^k - \sum_{\substack{j\in H_R:\\(i,j)\in E_R}} x_{i,j}^k = 0 \qquad \forall\, i\in N_R, k\in K \tag{2e}$$

$$1 - \sum_{j\in N_R}\sum_{\substack{i\in N_R:\\(i,j)\in E_R}} x_{i,j}^k \le M_3\cdot x_{\delta_{\text{start}},\delta_{\text{end}}}^k \qquad \forall\, k\in K \tag{2f}$$

$$B_j \ge (B_{\delta_{\text{start}}}^k + b_{\delta_{\text{start}}} + t_{\delta_{\text{start}},j})\cdot x_{\delta_{\text{start}},j}^k \qquad \forall\, j\in P\cup\bar{P}\cup\{\delta_{\text{end}}\}, k\in K \tag{2g}$$

$$B_{\delta_{\text{end}}}^k \ge (B_i + b_i + t_{i,\delta_{\text{end}}})\cdot x_{i,\delta_{\text{end}}}^k \qquad \forall\, i\in D\cup\bar{D}\cup\{\delta_{\text{start}}\}, k\in K \tag{2h}$$

$$B_j \ge (B_i + b_i + t_{i,j})\cdot\sum_{k\in K} x_{i,j}^k \qquad \forall\, i,j\in N_R : (i,j)\in E_R \tag{2i}$$

$$B_i \ge e_i + \sum_{j\in H_R\setminus\{i\}}\left(\max\{0, e_j - e_i + b_j + t_{j,i}\}\cdot\sum_{k\in K} x_{j,i}^k\right) \qquad \forall\, i\in N_R \tag{2j}$$

$$B_i \le l_i - \sum_{j\in H_R\setminus\{i\}}\left(\max\{0, l_i - l_j + b_i + t_{i,j}\}\cdot\sum_{k\in K} x_{i,j}^k\right) \qquad \forall\, i\in N_R \tag{2k}$$

$$L_i = B_{i+m} - (B_i + b_i) \qquad \forall\, i\in P \tag{2l}$$

$$t_{i,i+m} \le L_i \le \alpha \cdot t_{i,i+m} \qquad \forall\, i \in P \tag{2m}$$

$$Q_j \ge q_j \cdot x^k_{\delta_{\text{start}},j} \qquad \forall\, j \in P \cup \bar{P}, k \in K \tag{2n}$$

$$Q_j \ge (Q_i + q_j) \cdot \sum_{k \in K} x^k_{i,j} \qquad \forall\, i,j \in N_R : (i,j) \in E_R \tag{2o}$$

$$0 \ge Q_i \cdot x^k_{i,\delta_{\text{end}}} \qquad \forall\, i \in D \cup \bar{D}, k \in K \tag{2p}$$

$$Q_i \le Q_{\max} \cdot (1 - \sum_{k \in K} x^k_{i,j}) \qquad \forall\, (i,j) \in E_{\text{turn}} \tag{2q}$$

$$Q_i \ge -Q_{\max} \cdot (1 - \sum_{k \in K} x_{i,j}) \qquad \forall\, (i,j) \in E_{\text{turn}} \tag{2r}$$

$$Q_i \le Q_{\max} \cdot \sum_{k \in K} \sum_{\substack{j \in H_R: \\ (i,j) \in E_R}} x^k_{i,j} \qquad \forall\, i \in H_R \tag{2s}$$

$$z_k \ge \frac{1}{|H_R|^2} \cdot \sum_{(i,j) \in E_R} x^k_{i,j} \qquad \forall\, k \in K \tag{2t}$$

$$z_k \ge z_{k+1} \qquad \forall\, k \in K \tag{2u}$$

Constraints (2a) and (2b) ensure each passenger is picked up at most once and is dropped-off by the same vehicle. Vehicles must start (2c) at δ_{start} and end at δ_{end} (2d) depots, maintaining flow conservation across all arcs (2e). Only unused vehicles may use the arc $(\delta_{\text{start}}, \delta_{\text{end}})$, as denoted by (2f), where we use a big-M constraint with $M_3 := |E_R|$. The service start times for leaving and entering the depot, as well as consistency across arcs, is handled by (2g)–(2i). Constraints (2j) and (2k) ensure time consistency regarding the requests time windows. The maximum ride time of each request is defined and bounded by constraints (2l) and (2m). Load constraints (2n)–(2s) ensure vehicles respect capacity limits at each bus stop as well as on turning arcs. Constraint (2t) counts the number of required vehicles. Finally, we use the symmetry breaking constraint (2u) to improve computational times.

A Bi-Objective Optimization Model for Fare Structure Design in Public Transport

Philine Schiewe
Department of Mathematics and Systems Analysis, Aalto University, Finland

Anita Schöbel
Department of Mathematics, University of Kaiserslautern-Landau (RPTU), Germany
Fraunhofer Institute of Industrial Mathematics ITWM, Kaiserslautern, Germany

Reena Urban
Department of Mathematics, University of Kaiserslautern-Landau (RPTU), Germany

Abstract

Fare planning in public transport is important from the view of passengers as well as of operators. In this paper, we propose a bi-objective model that maximizes the revenue as well as the number of attracted passengers. The potential demand per origin-destination pair is divided into demand groups that have their own willingness how much to pay for using public transport, i.e., a demand group is only attracted as public transport passengers if the fare does not exceed their willingness to pay. We study the bi-objective problem for flat and distance tariffs and develop specialized algorithms to compute the Pareto front in quasilinear or cubic time, respectively. Through computational experiments on structured data sets we evaluate the running time of the developed algorithms in practice and analyze the number of non-dominated points and their respective efficient solutions.

2012 ACM Subject Classification Applied computing → Transportation

Keywords and phrases Public transport, fare structure design, modeling, bi-objective, algorithm

Digital Object Identifier 10.4230/OASIcs.ATMOS.2024.15

Funding This work was partially funded by the European Union's Horizon 2020 research and innovation programme [Grant 875022] and by the Federal Ministry of Education and Research [Project 01UV2152B] under the project sEAmless SustaInable EveRyday urban mobility (EASIER).

1 Introduction

Fare structures in public transport are an important design element that involves the interests of both (potential) passengers and operators alike. For passengers, fares are one among several criteria for mode and route choice. The affordability and the perceived fairness of fares significantly influence people's decisions to opt for public transport over other modes of transport, for example, their own car. When the fares exceed a certain price limit (willingness to pay), it is reasonable to assume a deterrent effect leading to a reduction in the attractiveness of public transport and, therefore, ridership. Conversely, for operators, fares directly impact the revenue. An increase of prices, for example, increases the income per sold ticket but might decrease the ridership and therefore the total number of sold tickets.

In this paper, we investigate the trade-off between revenue and number of passengers for different fare strategies. For each origin-destination (OD) pair, we consider multiple demand groups that differ in their willingness to pay. If the fare for an OD pair exceeds the willingness to pay of a demand group, this group does not use public transport. These demand groups could, for example, be captive and choice passengers, where the willingness to pay is dependent on whether an alternative mode like a car is available or not. Another categorization of demand groups could be based on age and income. We introduce a bi-objective model that optimizes fare structures to determine the Pareto front of revenue and number of passengers.

24th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2024).
Editors: Paul C. Bouman and Spyros C. Kontogiannis; Article No. 15; pp. 15:1–15:19
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

A wide range of fare structures is implemented worldwide. In this paper, we introduce the revenue-passenger model for fare structures in general and then focus on flat and distance tariffs. In a flat tariff, all tickets have the same price. While this is very easy to understand, it also encourages using public transport for longer journeys. For shorter journeys a flat tariff might be perceived as unfair because passengers with a short journey pay the same price as passengers with a long journey. The opposite can be realized with a distance tariff, which accounts for the traveled distance. Here, the distance may either be the beeline distance between the start and the end station of the journey or the network distance of the respective path of the journey. In this paper, we consider affine distance tariffs, which are composed of a base amount and an additional price per kilometer. While it is easy to communicate, passengers need to know the exact distance of their journey to determine the fare. Slight deviations of the path may directly lead to a change in the fare. Other differential fare structures depend on a duration, time or quality component of the journey [12, 22, 9], but are not considered here.

Related Literature. In public transport, requirements and needs of several actors such as the passengers and the operators are involved, leading to multi-objective models [5]. However, multi-objective models are scarcely considered in fare planning, and the literature on fare planning so far focuses on single-objective models. The objective often is the minimization of the deviation from reference prices [15, 16, 1] or the maximization of either revenue or demand [10, 3, 26, 19, 17]. Moreover, studies analyze the impact of different fare structures such as flat, distance and zone tariffs on the route choice and travel time [18] and the revenue and number of passengers [13, 6]. We expand the literature by a bi-objective model with respect to revenue and number of passengers.

Contribution. First, we formulate a general model that can be applied for any fare strategy. Due to the characteristic of one of the objectives, the complete Pareto front can be determined with the ϵ-constraint method. Second, we study the specific problem for flat and distance tariffs. In both cases, we identify a finite candidate set, based on which we develop algorithms that compute the Pareto front in quasilinear or cubic time, respectively.

We also perform computational experiments on structured data sets and analyze the number of non-dominated points and their respective efficient solutions. The experiments emphasize the advantage in running time of the specialized algorithm for distance tariffs compared to the mixed-integer programming formulation derived from the general model.

2 Problem Formulation

Let a *public transport network (PTN)* (V, E) be given. The node set V represents a set of stops or stations and the edge set E represents the direct connections between them. For simplicity, we assume the PTN to be an undirected graph which is simple and connected. The PTN can be used to model railway, tram, or bus networks. In the following, we call the nodes of the PTN stations, also if bus networks with stops are under consideration.

By $D \subseteq \{(v_1, v_2) : v_1, v_2 \in V, v_1 \neq v_2\}$ we denote the set of origin-destination (OD) pairs. The potential passengers of an OD pair can be distinguished by their willingness to pay. This could for example reflect the degree of dependence on public transport or the income. For each OD pair $d \in D$, we denote by $G_d \in \mathbb{N}_{\geq 1}$ the number of demand groups, by $t_d^g \in \mathbb{N}_{\geq 1}$ the number of people belonging to group $g \in \{1, \ldots, G_d\}$ with a *willingness to pay* of $w_d^g \in \mathbb{R}_{>0}$. In this model, a demand group uses public transport whenever the ticket price does not exceed its willingness to pay. Without loss of generality, we assume that $w_d^g > 0$ and $w_d^g \neq w_d^{g'}$ for all $g, g' \in \{1, \ldots, G_d\}$, $g \neq g'$ and all OD pairs $d \in D$.

To simplify notation, we introduce the shorthand notation $[G] := \{1, \dots, G\}$ for $G \in \mathbb{N}_{\geq 1}$.

▶ **Definition 1** (Fare structure [25]). *Let a PTN be given, and let $\mathcal{W}$ be the set of all paths in the PTN. A* fare structure *is a function $\pi\colon \mathcal{W} \to \mathbb{R}_{\geq 0}$ that assigns a price to every path in the PTN.*

Usually, a fare strategy (e.g., a flat, distance or zone tariff) is desired instead of just determining a price for each OD pair. Such a desired fare strategy can be modeled by additional requirements, that can be formulated as constraints.

The objective is to maximize the revenue and the number of passengers simultaneously. While the revenue is the key objective of the operator, the number of passengers serves as an indicator of the success of the transition towards sustainable transport modes. This is particularly significant when public transport is used instead of private motorized transport modes such that the environmental impact of traveling is reduced.

Given a fare structure π, we denote the according price for OD pair $d \in D$ by π_d. The number of attracted passengers for OD pair $d \in D$ given π_d is then determined as

$$\mathsf{pass}(d \mid \pi_d) := \sum_{g \in [G_d]\colon \pi_d \leq w_d^g} t_d^g.$$

The total number of passengers with respect to fare structure π is

$$\mathsf{pass}(\pi) := \sum_{d \in D} \mathsf{pass}(d \mid \pi_d)$$

and the total revenue is

$$\mathsf{rev}(\pi) := \sum_{d \in D} \mathsf{pass}(d \mid \pi_d) \cdot \pi_d.$$

With this, we can now define the revenue-passenger model formally.

▶ **Definition 2** (The revenue-passenger model). *Given are*

- *a PTN (V, E) as an undirected graph,*
- *a set of OD pairs $D \subseteq \{(v_1, v_2) : v_1, v_2 \in V, v_1 \neq v_2\}$, $D \neq \emptyset$,*
- *the numbers of demand groups $G_d \in \mathbb{N}_{\geq 1}$ for each OD pair $d \in D$,*
- *the willingness to pay $w_d^g \in \mathbb{R}_{\geq 0}$ and the potential demand $t_d^g \in \mathbb{N}_{\geq 1}$ for each demand group $g \in [G_d]$ and each OD pair $d \in D$.*

The aim is to determine fare structures π that maximize the revenue $\mathsf{rev}(\pi)$ *and the number of passengers* $\mathsf{pass}(\pi)$*, where a desired fare strategy might be required. The bi-objective problem hence is:*

$$\begin{aligned} \max\ & \mathsf{rev}(\pi) \\ \max\ & \mathsf{pass}(\pi) \\ \text{s.t.}\ & \pi \text{ is of a desired fare strategy} \\ & \pi_d \geq 0 \quad \text{for all } d \in D. \end{aligned}$$

Each feasible fare structure π induces a two-dimensional vector of objective function values, i.e., we have a bi-objective problem. For multi-objective optimization, we refer to, e.g., [11]. As usual in multi-objective optimization, we are interested in finding the Pareto front and corresponding efficient solutions. Generally speaking, we aim to find those feasible fare structures that do not allow to improve one objective function without decreasing the other.

▶ **Definition 3** (Efficient solution, non-dominated point and Pareto front, e.g., [11]). *Let an instance of the* revenue-passenger model *be given. A feasible solution π is called* efficient *and its objective value* $(\mathsf{rev}(\pi), \mathsf{pass}(\pi))$ *is called* non-dominated *if there does not exist another feasible solution π' with objective value* $(\mathsf{rev}(\pi'), \mathsf{pass}(\pi'))$ *such that* $\mathsf{rev}(\pi') \geq \mathsf{rev}(\pi)$ *and* $\mathsf{pass}(\pi') \geq \mathsf{pass}(\pi)$ *and at least one inequality holding strictly. The set of all non-dominated points is also called the* Pareto front*.*

Because the numbers of passengers t_d^g for all $d \in D$, $g \in \{1, \ldots, G_d\}$ are given as natural numbers, the objective function pass always attains integral values. Hence, the whole Pareto front can be computed systematically by applying the well-known ϵ-constraint method [11, 4]. This is done by restricting the number of passengers pass in the constraints while the revenue rev remains as the objective function. In this case, by increasing ϵ with a step width of 1, we do not miss any non-dominated point. Further, given a set of tuples of revenue and number of passengers, it is simple to *filter for the non-dominated points*: The points are first sorted in decreasing order by the number of passengers and as a second criterion by decreasing revenue. We then iterate over this sorted list and add a point to the Pareto front whenever the revenue is strictly higher than the highest value so far. This ensures that while the numbers of passengers is decreasing, the revenue is increasing, and we only keep non-dominated points.

3 Flat Tariffs

Flat tariffs are common in city centers and assign the same price to all paths. We start with a formal definition:

▶ **Definition 4** (Flat tariff, [25]). *Let a PTN be given, and let $\mathcal{W}$ be the set of all paths in the PTN. A fare structure π is a* flat tariff *w.r.t. a fixed price $f \in \mathbb{R}_{\geq 0}$ if $\pi(W) = f$ for all $W \in \mathcal{W}$.*

Let $S_{\text{will}} := \{w_d^g : d \in D, g \in [G_d]\}$ be the set of all willingness to pay values with $\max S_{\text{will}}$ the largest of these values and let

$$S := \left\{ (w,t) : w \in S_{\text{will}}, t = \sum_{d \in D} \sum_{g \in [G_d] :\, w_d^g = w} t_d^g \right\}$$

be the set of all tuples of willingness to pay and the respective demand with exactly this willingness to pay. Let $(w_1, t_1), \ldots, (w_{|S|}, t_{|S|})$ be a sorting of S such that $w_1 < \ldots < w_{|S|}$. In particular, we have $|S| \leq \sum_{d \in D} G_d$, with equality if and only if the willingness to pay is different for every demand group.

For a flat tariff π with fixed price $f \in \mathbb{R}_{\geq 0}$, the objective functions simplify to

$$\mathsf{rev}(\pi) = f \cdot \sum_{\substack{(w,t) \in S: \\ f \leq w}} t \quad \text{and} \quad \mathsf{pass}(\pi) = \sum_{\substack{(w,t) \in S: \\ f \leq w}} t.$$

Because a flat tariff π is uniquely determined by f, we write $\mathsf{rev}(f)$ and $\mathsf{pass}(f)$ instead of $\mathsf{rev}(\pi)$ and $\mathsf{pass}(\pi)$.

▶ **Definition 5** (F-RPM)**.** *Given the input data as in Definition 2, the bi-objective revenue-passenger model for a flat tariff (*F-RPM*) is the following:*

$$\begin{aligned} \max\ \mathsf{rev}(f) &= f \cdot \sum_{\substack{(w,t)\in S:\\ f\le w}} t \\ \max\ \mathsf{pass}(f) &= \sum_{\substack{(w,t)\in S:\\ f\le w}} t \\ \text{s.t.}\quad & f \in \mathbb{R}_{\ge 0}. \end{aligned}$$

We now derive a finite candidate set for F-RPM.

▶ **Lemma 6.** *For all efficient solutions f to* F-RPM*, it holds that $f \in S_{\text{will}}$.*

Proof. Let $\bar{f}$ be an efficient solution, and assume that $\bar{f} \notin S_{\text{will}}$. First, we have that $\bar{f} < \max S_{\text{will}}$ because for $\bar{f} > \max S_{\text{will}}$ the objective function values are $(0, 0)$, which is not efficient since for $f := \max S_{\text{will}} = w_{|S|}$ the objective function values are $(\mathsf{rev}(f), \mathsf{pass}(f)) = (t_{|S|} \cdot w_{|S|}, t_{|S|})$, which dominates $(0, 0)$. Hence, $f' := \min\{w \in S_{\text{will}} : \bar{f} < w\}$ is well-defined and f' is the next higher price compared to $\bar{f}$ that is contained in S_{will}. Then $\bar{f} < f'$ and $\{(w, t) \in S : \bar{f} \le w\} = \{(w, t) \in S : f' \le w\}$ by definition of f' and because $\bar{f} \notin S_{\text{will}}$. This yields $\mathsf{pass}(\bar{f}) = \mathsf{pass}(f')$ and $\mathsf{rev}(\bar{f}) = \bar{f} \cdot \mathsf{pass}(\bar{f}) < f' \cdot \mathsf{pass}(\bar{f}) = \mathsf{rev}(f')$, which is a contradiction to $\bar{f}$ being efficient. ◀

Lemma 6 allows Algorithm 1 to compute the Pareto front in $\mathcal{O}(|S| \cdot \log(|S|))$. Note that $|S| \le \sum_{d\in D} G_d$.

Algorithm 1 Solution method for F-RPM.

```
   Input  : Set $S$ (as defined above) of F-RPM
   Output : Set $\Gamma$ of all non-dominated points
1  Sort $S$ such that $w_1 < \ldots < w_{|S|}$.
2  Initialize $\overline{\mathsf{pass}} \leftarrow \sum_{s=1}^{|S|} t_s$;  $\overline{\mathsf{rev}} \leftarrow w_1 \cdot \overline{\mathsf{pass}}$;  $\Gamma \leftarrow \{(\overline{\mathsf{rev}}, \overline{\mathsf{pass}})\}$;  $\mathsf{rev}^* \leftarrow \overline{\mathsf{rev}}$.
3  for $s = 2, \ldots, |S|$ do
4      Update $\overline{\mathsf{pass}} \leftarrow \overline{\mathsf{pass}} - t_{s-1}$.
5      Update $\overline{\mathsf{rev}} \leftarrow w_s \cdot \overline{\mathsf{pass}}$.
6      if $\overline{\mathsf{rev}} > \mathsf{rev}^*$ then
7          Update $\Gamma \leftarrow \Gamma \cup \{(\overline{\mathsf{rev}}, \overline{\mathsf{pass}})\}$.
8          Update $\mathsf{rev}^* \leftarrow \overline{\mathsf{rev}}$.
9  return $\Gamma$
```

▶ **Theorem 7.** *Algorithm 1 solves* F-RPM *in $\mathcal{O}(|S| \cdot \log(|S|))$.*

Proof. By Lemma 6 it suffices to consider the willingness to pay $w_s \in S_{\text{will}}$ as fixed prices of the flat tariff. Because w_1 is the unique optimum with respect to the objective function pass, $(\mathsf{rev}(w_1), \mathsf{pass}(w_1))$ is a non-dominated point and is added to Γ in line 2. In rev^* we store the maximum revenue that has occurred so far. Increasing the fixed price from w_{s-1} to w_s reduces the number of passengers by those that have a willingness to pay of w_{s-1}, which are t_{s-1} many. Hence, after the updates in lines 4 and 5, $\overline{\mathsf{rev}}$ and $\overline{\mathsf{pass}}$ are the revenue and the number of passengers for a flat tariff with fixed price w_s. Because the number of passengers is strictly decreased in every iteration, the tuple $(\overline{\mathsf{rev}}, \overline{\mathsf{pass}})$ is non-dominated whenever the revenue $\overline{\mathsf{rev}}$ is larger than any previous revenue, i.e., if $\overline{\mathsf{rev}} > \mathsf{rev}^*$. Therefore, in this case, the tuple is added to Γ and the maximum revenue rev^* is updated.

Sorting S can be done in $\mathcal{O}(|S| \cdot \log(|S|))$ (see, e.g., [7]). The initialization of $\overline{\mathsf{pass}}$ in line 2 is executed in $\mathcal{O}(|S|)$, whereas all other initializations and updates are in $\mathcal{O}(1)$. Hence, the for-loop takes $\mathcal{O}(|S|)$ in total. Overall, we obtain a running time of $\mathcal{O}(|S| \cdot \log(|S|))$. ◀

4 Distance Tariffs

In a distance tariff, the fare is related to a distance $l(W)$ associated with the path $W \in \mathcal{W}$. Usually this is the beeline (Euclidean) distance between the start and the end station of the path or the network distance, i.e., the length of the path W in the PTN. We consider affine distance tariffs which means that the fares consist of a base amount and a price per kilometer. We start with a formal definition.

▶ **Definition 8** (Affine distance tariff, [25]). *Let a PTN be given, and let $\mathcal{W}$ be the set of all paths in the PTN. Let $l \colon \mathcal{W} \to \mathbb{R}_{\geq 0}$ be a distance function determining, e.g., the beeline or network distance. A fare structure π is an* affine distance tariff *w.r.t. a base amount $f \in \mathbb{R}_{\geq 0}$ and a price per kilometer $p \in \mathbb{R}_{\geq 0}$ if $\pi(W) = f + p \cdot l(W)$ for all $W \in \mathcal{W}$.*

For the optimization of affine distance tariffs, we consider that each OD pair $d \in D$ travels along a fixed path $W_d \in \mathcal{W}$. Hence, each OD pair is associated with a distance $l_d := l(W_d)$ based on the distance function l.

▶ **Definition 9** (D-RPM). *Given the input data as in Definition 2 and a distance $l_d \in \mathbb{R}_{\geq 0}$ associated with OD pair d for all $d \in D$, the bi-objective revenue-passenger model for a distance tariff (*D-RPM*) is the following:*

$$\begin{aligned}
\max \quad & \mathsf{rev}(\pi) \\
\max \quad & \mathsf{pass}(\pi) \\
\text{s.t.} \quad & \pi_d = f + p \cdot l_d \qquad \text{for all } d \in D \\
& f, p \in \mathbb{R}_{\geq 0}.
\end{aligned}$$

Also here, we write $\mathsf{rev}(f,p)$ and $\mathsf{pass}(f,p)$ instead of $\mathsf{rev}(\pi)$ and $\mathsf{pass}(\pi)$ because an affine distance tariff is uniquely determined by f and p.

For the ϵ-constraint method, the following mixed-integer linear programming (MILP) formulation may be used:

$$\begin{aligned}
\max_{f,p,\pi_d^g,x_d^g} \quad & \sum_{d \in D} \sum_{g \in [G_d]} t_d^g \cdot \pi_d^g \\
\text{s.t.} \quad & \epsilon \leq \sum_{d \in D} \sum_{g \in [G_d]} t_d^g \cdot x_d^g \\
& f + p \cdot l_d \leq w_d^g + M \cdot (1 - x_d^g) && \text{for all } d \in D, g \in [G_d] && (1) \\
& \pi_d^g \leq f + p \cdot l_d && \text{for all } d \in D, g \in [G_d] && (2) \\
& \pi_d^g \leq M \cdot x_d^g && \text{for all } d \in D, g \in [G_d] && (3) \\
& f, p, \pi_d^g \in \mathbb{R}_{\geq 0} && \text{for all } d \in D, g \in [G_d] \\
& x_d^g \in \{0,1\} && \text{for all } d \in D, g \in [G_d].
\end{aligned}$$

The variables f and p determine the base amount and the price per kilometer of the distance tariff. The binary variable x_d^g is 1 if and only if demand group g of OD pair d uses public transport. Finally, the variable π_d^g stores the price that is actually paid by the demand group g of OD pair d. Constraints (1) ensure that x_d^g is set correctly, meaning that it is only 1 if the price according to the distance tariff does not exceed the willingness to pay. Constraints (2) limit the price of a demand group to the price of the distance tariff and constraints (3) set the price paid by a demand group to 0 if it does not use public transport. Together constraints (2) and (3) set the price paid by a demand group to either 0 or the distance tariff price.

Before we show that M can be chosen based on natural bounds on f and p, we introduce some notation. Let

$$S_{\text{dem}}(f,p) := \{(d,g) : d \in D, g \in [G_d], w_d^g \geq f + p \cdot l_d\}$$

be the set of demand groups that are attracted in case of a distance tariff with base amount f and price per kilometer p, i.e., $\mathsf{pass}(f,p) = \sum_{(d,g)\in S_{\text{dem}}(f,p)} t_d^g$. For every efficient solution (f,p), it holds that $S_{\text{dem}}(f,p) \neq \emptyset$ because otherwise the objective function value is $(0,0)$ and is, analogously to the proof of Lemma 6, dominated by a solution $f' := w_{\bar{d}}^{\bar{g}}$ for some $\bar{d} \in D$, $\bar{g} \in [G_{\bar{d}}]$ and $p' := 0$, which attracts at least one demand group. Therefore, we can restrict $f \leq f^{\max} := \max\{w_d^g : d \in D, g \in [G_d]\}$ and $p \leq p^{\max} := \max\left\{\frac{w_d^g}{l_d} : d \in D, g \in [G_d]\right\}$. Let $l^{\max} := \max\{l_d : d \in D\}$. Setting $M := f^{\max} + p^{\max} \cdot l^{\max}$, we have for all $d \in D$, $g \in [G_d]$ that

$$\pi_d^g \overset{(2)}{\leq} f + p \cdot l_d \leq f^{\max} + p^{\max} \cdot l^{\max} = M,$$

i.e., M is sufficiently large for constraints (1) and (3).

This MILP has $\mathcal{O}(\sum_{d\in D} G_d)$ many variables and constraints and, as we will see later in the experiments, is hard to solve. We hence develop a polynomial-time method that exploits the specific problem structure.

▶ **Lemma 10.** *For every efficient solution (f,p), at least one willingness to pay is met exactly, i.e., there is at least one willingness to pay w_d^g for some OD pair $d \in D$ and demand group $g \in [G_d]$ such that $w_d^g = f + p \cdot l_d$.*

Proof. Let $(\bar{f},\bar{p})$ be an efficient solution. Assume that no willingness to pay is met exactly. Then there must be a willingness to pay w_d^g for some $d \in D$, $g \in [G_d]$ such that $w_d^g > \bar{f} + \bar{p} \cdot l_d$. Otherwise $S_{\text{dem}}(\bar{f},\bar{p}) = \emptyset$, which is not possible for an efficient solution. Hence, we can set $\delta := \min\{w_d^g - \bar{f} - \bar{p} \cdot l_d > 0 : d \in D, g \in [G_d]\}$. Increasing $\bar{f}$ to $f' := \bar{f} + \delta$, we have

$$\{(d,g) : d \in D, g \in [G_d], \bar{f} + \bar{p} \cdot l_d \leq w_d^g\} = \{(d,g) : d \in D, g \in [G_d], f' + \bar{p} \cdot l_d \leq w_d^g\}$$

and hence $\mathsf{pass}(\bar{f},\bar{p}) = \mathsf{pass}(f',\bar{p})$ and $\mathsf{rev}(\bar{f},\bar{p}) < \mathsf{rev}(\bar{f},\bar{p}) + \delta \cdot \mathsf{pass}(\bar{f},\bar{p}) = \mathsf{rev}(f',\bar{p})$ which is a contradiction to $(\bar{f},\bar{p})$ being an efficient solution. ◀

Lemma 10 shows that we can interpret our problem as least absolute deviation (LAD) regression problem (see, e.g., [2, 23, 8, 24]): Given a set of points with weights, find a line that minimizes the sum of vertical distances to the given points. In our case, the existing points are (l_d, w_d^g) with weights t_d^g for every OD pair $d \in D$ and every demand group $g \in [G_d]$. We want to fit these points by a line with intercept f and slope p, i.e., $x \mapsto f + p \cdot x$. For the evaluation of the fit we distinguish two cases:

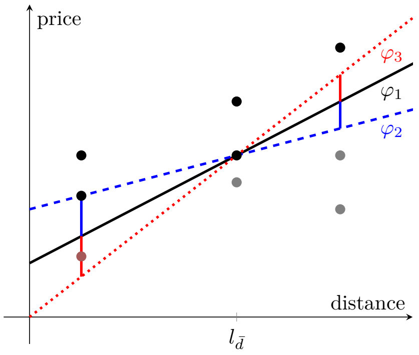


Figure 1 Example of three OD pairs with distances 1, 4 and 6 and with three groups each. They are marked based on their distance and willingness to pay. Line φ_1 (black, solid) meets exactly one willingness to pay exactly. The point $(l_{\bar{d}}, w_{\bar{d}}^{\bar{g}})$ is the fixed point. The line can be rotated clockwise to line φ_2 (blue, dashed), or it can be rotated counterclockwise to line φ_3 (red, dotted). For φ_2 the willingness to pay of two groups is met exactly. The black groups are always attracted, the gray are attracted in none of the scenarios, and the gray-red group is only attracted in case of φ_3 (red, dotted). Vertical lines show the price difference for each OD pair in the different scenarios.

- If (l_d, w_d^g) is on or above the line, we add t_d^g to $\mathsf{pass}(f, p)$. For $\mathsf{rev}(f, p)$, we could have achieved the full amount of the willingness to pay w_d^g, but we realize only the point on the line, i.e., $f + p \cdot l_d$. The vertical distance $w_d^g - (f + p \cdot l_d)$ between the point (l_d, w_d^g) and the line is what we lose and hence what we want to minimize.
- If (l_d, w_d^g) is below the line, the OD pair is lost and hence does not contribute to any of the two objective functions.

Lemma 10 then says that any optimal line passes through at least one of the points. For unrestricted LAD lines it is furthermore known that there always exists an optimal line that passes through two of the points. In our case, the parameters of the line are restricted to be positive, i.e., $f \geq 0$ and $p \geq 0$. Taking this restriction into account leads to the statement of Theorem 11.

▶ **Theorem 11.** *For every non-dominated point, there is an efficient solution (f, p) such that one of the following holds: The willingness to pay of*

- *two groups is met exactly, i.e., there are $d_i \in D$, $g_i \in [G_{d_i}]$ for $i \in \{1, 2\}$ with $d_1 \neq d_2$ and $w_{d_i}^{g_i} = f + p \cdot l_{d_i}$,*
- *one group is met exactly and $p = 0$, i.e., there is some $d \in D$, $g \in [G_d]$ with $w_d^g = f$,*
- *one group is met exactly and $f = 0$, i.e., there is some $d \in D$, $g \in [G_d]$ with $w_d^g = p \cdot l_d$.*

Proof. Let (f_1, p_1) be an efficient solution, in particular $f_1, p_1 \in \mathbb{R}_{\geq 0}$. By Lemma 10, there is at least one willingness to pay that is met exactly. We consider the case that only exactly one willingness to pay $\bar{w} := w_{\bar{d}}^{\bar{g}}$ for some $\bar{d} \in D$, $\bar{g} \in [G_{\bar{d}}]$ is met exactly and that neither $p_1 = 0$ nor $f_1 = 0$. We fix $(\bar{w}, l_{\bar{d}})$ and rotate the line $f_1 + p_1 \cdot x$ in the following ways as illustrated in Figure 1:

We choose (f_2, p_2) as the optimal solution to

$$\begin{aligned}
&\min_{f,p} \; p \\
&\text{s.t. } \bar{w} = f + p \cdot l_{\bar{d}} \\
&\qquad S_{\text{dem}}(f_1, p_1) \subseteq S_{\text{dem}}(f, p) \\
&\qquad p_1 \geq p \geq 0 \\
&\qquad f \geq 0.
\end{aligned}$$

Hence, (f_2, p_2) still meets $\bar{w}$ for OD pair $\bar{d}$. The line $f_2 + p_2 \cdot x$ is less steep than $f_1 + \bar{p}_1 \cdot x$, with a non-negative slope for which all demand groups that are attracted by (f_1, p_1) are also attracted by (f_2, p_2).

Analogously, we choose (f_3, p_3) as the optimal solution to

$$\begin{aligned} &\max_{f,p} p \\ &\text{s.t.}\ \ \bar{w} = f + p \cdot l_{\bar{d}} \\ &\qquad S_{\text{dem}}(f_1, p_1) \subseteq S_{\text{dem}}(f, p) \\ &\qquad p \geq p_1 \\ &\qquad f \geq 0. \end{aligned}$$

Note that (f_2, p_2) and (f_3, p_3) are of the form as in the claim. Because of the assumption that only one willingness to pay is met exactly and $p_1 > 0$ and $f_1 > 0$, we have that $p_2 < p_1 < p_3$ and $f_3 < f_1 < f_2$. For $i \in \{1, 2, 3\}$, we define $\varphi_i \colon \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0},\ x \mapsto f_i + p_i \cdot x$.

By construction, we ensure that the numbers of attracted passengers are not decreased when changing from (f_1, p_1) to (f_2, p_2) or to (f_3, p_3). We show that for at least one of these options also the revenue does not decrease. Note that the revenue related to OD pairs with distance $l_{\bar{d}}$ does not change because the price is kept fixed. Hence, we divide the attracted demand groups based on their respective distances l_d compared to $l_{\bar{d}}$ as follows:

$$S_{\text{dem},L} := \{(d, g) \in S_{\text{dem}}(f_1, p_1) : l_d < l_{\bar{d}}\} \quad \text{and} \quad S_{\text{dem},R} := \{(d, g) \in S_{\text{dem}}(f_1, p_1) : l_d > l_{\bar{d}}\}.$$

For all $d \in D$, we set $\delta_d := |l_{\bar{d}} - l_d|$. From that, we obtain that $l_d = l_{\bar{d}} - \delta_d$ for $(d, g) \in S_{\text{dem},L}$ and $l_d = l_{\bar{d}} + \delta_d$ for $(d, g) \in S_{\text{dem},R}$, which yields for all $i \in \{1, 2, 3\}$ and all $d \in D$ that

$$\varphi_i(l_d) = f_i + p_i \cdot (l_{\bar{d}} \mp \delta_d) = f_i + p_i \cdot l_{\bar{d}} \mp p_i \cdot \delta_d = \varphi_i(l_{\bar{d}}) \mp p_i \cdot \delta_d.$$

Then for the difference in revenues, we have

$$\begin{aligned} \mathsf{rev}(f_2, p_2) - \mathsf{rev}(f_1, p_1) \geq &\sum_{(d,g) \in S_{\text{dem},L}} t_d^g \left(\varphi_2(l_{\bar{d}}) - p_2\delta_d\right) + \sum_{(d,g) \in S_{\text{dem},R}} t_d^g \left(\varphi_2(l_{\bar{d}}) + p_2\delta_d\right) \\ &- \sum_{(d,g) \in S_{\text{dem},L}} t_d^g \left(\varphi_1(l_{\bar{d}}) - p_1\delta_d\right) - \sum_{(d,g) \in S_{\text{dem},R}} t_d^g \left(\varphi_1(l_{\bar{d}}) + p_1\delta_d\right) \\ = &\sum_{(d,g) \in S_{\text{dem},L}} t_d^g \cdot \delta_d \cdot (p_1 - p_2) + \sum_{(d,g) \in S_{\text{dem},R}} t_d^g \cdot \delta_d \cdot (p_2 - p_1) \\ = &\underbrace{(p_1 - p_2)}_{>0} \underbrace{\left(\sum_{(d,g) \in S_{\text{dem},L}} t_d^g \cdot \delta_d - \sum_{(d,g) \in S_{\text{dem},R}} t_d^g \cdot \delta_d \right)}_{:=\Delta} \end{aligned}$$

and analogously

$$\begin{aligned} \mathsf{rev}(f_3, p_3) - \mathsf{rev}(f_1, p_1) \geq &\sum_{(d,g) \in S_{\text{dem},L}} t_d^g \left(\varphi_3(l_{\bar{d}}) - p_3\delta_d\right) + \sum_{(d,g) \in S_{\text{dem},R}} t_d^g \left(\varphi_3(l_{\bar{d}}) + p_3\delta_d\right) \\ &- \sum_{(d,g) \in S_{\text{dem},L}} t_d^g \left(\varphi_1(l_{\bar{d}}) - p_1\delta_d\right) - \sum_{(d,g) \in S_{\text{dem},R}} t_d^g \left(\varphi_1(l_{\bar{d}}) + p_1\delta_d\right) \\ = &\sum_{(d,g) \in S_{\text{dem},L}} t_d^g \cdot \delta_d \cdot (p_1 - p_3) + \sum_{(d,g) \in S_{\text{dem},R}} t_d^g \cdot \delta_d \cdot (p_3 - p_1) \\ = &\underbrace{(p_3 - p_1)}_{>0} \underbrace{\left(\sum_{(d,g) \in S_{\text{dem},R}} t_d^g \cdot \delta_d - \sum_{(d,g) \in S_{\text{dem},L}} t_d^g \cdot \delta_d \right)}_{=-\Delta} \end{aligned}$$

Because $\Delta \geq 0$ or $-\Delta \geq 0$, at least one difference is non-negative. If the absolute revenue difference $|\Delta|$ is strictly larger than zero, then (f_1, p_1) is dominated by the respective other distance tariff, which is a contradiction. If it is equal to zero, but the number of passengers has increased, then (f_1, p_1) is again dominated, which is a contradiction. And if it is equal to zero and the number of passengers is the same, then all solutions belong to the same non-dominated point and we can choose the one that satisfies one of the claimed criteria.

Hence, for every non-dominated point, there is an efficient solution satisfying one of the criteria in the claim. ◀

▶ **Corollary 12.** *The problem is tractable, i.e., the number of non-dominated points is polynomial in the input, namely in $\mathcal{O}((\sum_{d\in D} G_d)^2)$.*

Proof. The claim follows from Theorem 11 because there are at most $\sum_{d\in D} G_d$ non-dominated points for efficient solutions (f, p) with $f = 0$ or $p = 0$, and at most $(\sum_{d\in D} G_d)^2$ non-dominated points that meet the willingness to pay of two demand groups exactly. ◀

Algorithm 2 Solution method for D-RPM.

```
Input  : Instance of D-RPM
Output : Set $\Gamma$ of all non-dominated points
Initialize $\Gamma_1 \leftarrow \emptyset$;  $\Gamma_2 \leftarrow \emptyset$;  $\Gamma_3 \leftarrow \emptyset$.
// Determine points with a solution with $p = 0$.
Apply Algorithm 1 and let $\Gamma_1$ be its result.
// Determine points with a solution with $f = 0$.
Set $f \leftarrow 0$.
for $d \in D$ do
    for $g \in [G_d]$ do
        Set $p \leftarrow \frac{w_d^g}{l_d}$.
        Update $\Gamma_2 \leftarrow \Gamma_2 \cup \{(\mathsf{rev}(f,p), \mathsf{pass}(f,p))\}$.
// Determine points with a solution that meets the willingness to pay
   of two groups exactly.
for $d, d' \in D$ with $l_d < l_{d'}$ do
    for $g \in [G_d]$, $g' \in [G_{d'}]$ do
        Set $p \leftarrow \frac{w_{d'}^{g'} - w_d^g}{l_{d'} - l_d}$.
        Set $f \leftarrow w_d^g - p \cdot l_d$.
        if $f > 0$ and $p > 0$ then
            Update $\Gamma_3 \leftarrow \Gamma_3 \cup \{(\mathsf{rev}(f,p), \mathsf{pass}(f,p))\}$.
// Filter for non-dominated points.
Filter $\Gamma_1 \cup \Gamma_2 \cup \Gamma_3$ for non-dominated solution as described in Section 2. Let $\Gamma$ be the
  filtered result.
return $\Gamma$
```

▶ **Theorem 13.** *Algorithm 2 computes the set of all non-dominated points of* D-RPM *in $\mathcal{O}((\sum_{d\in D} G_d)^3)$.*

Proof. Theorem 11 gives a characterization of efficient solutions from which all non-dominated solutions can be determined. In line 2 of Algorithm 2, a superset of all non-dominated solutions with an efficient solution (f, p) with $p = 0$ is determined, in lines 3 to 7 a superset of all those with $f = 0$ and in lines 8 to 13 of all those that meet the willingness to pay of at least two groups exactly are computed. Combinations of demand groups with the same distance are omitted because this would yield an infeasible vertical line. Therefore, $\Gamma_1 \cup \Gamma_2 \cup \Gamma_3$ contains all non-dominated points. In line 14, all dominated solutions are removed and, hence, Γ is the set of all non-dominated points.

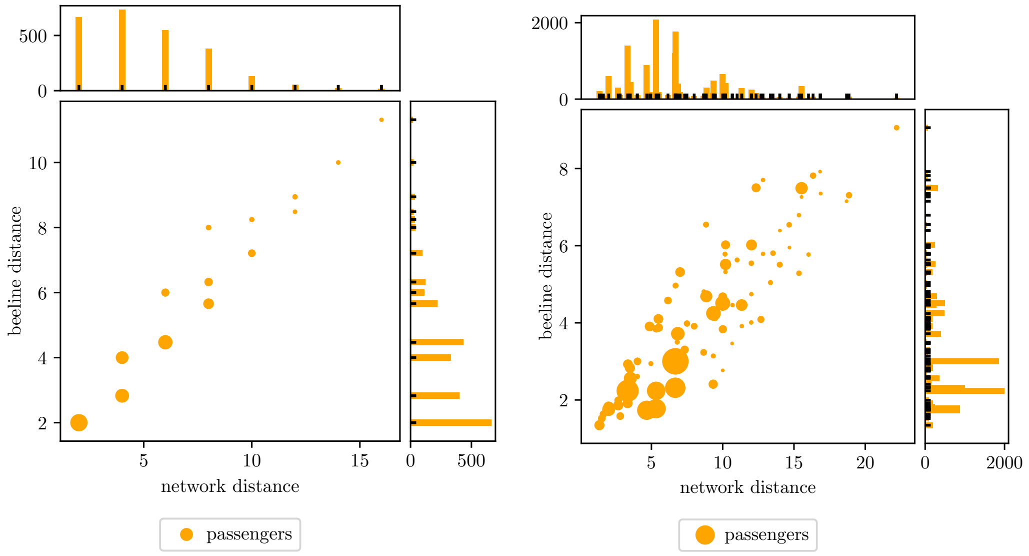
The computations in lines 2 to 7 are in $\mathcal{O}((\sum_{d \in D} G_d)^2)$. In lines 8 to 13, we iterate over the combinations of two demand groups and again iterate over the demand groups for determining the revenue and the number of passengers in line 13. This is done in $\mathcal{O}((\sum_{d \in D} G_d)^3)$. Filtering $\Gamma_1 \cup \Gamma_2 \cup \Gamma_3$ for non-dominated solutions in line 14 is done in $\mathcal{O}((\sum_{d \in D} G_d)^2 \cdot \log(\sum_{d \in D} G_d))$. Hence, in total, the algorithm is in $\mathcal{O}((\sum_{d \in D} G_d)^3)$. ◀

▶ Remark 14. Note that the running time is significantly influenced by the number of OD pairs with the same distance because the for-loop in line 8 of Algorithm 2 is only performed for OD pairs d' with a larger distance than that of OD pair d, but in particular not for those with the same distance. Hence, the loops over the demand groups and the computation of the objective function value are omitted for OD pair combinations with the same distance.

5 Computational Experiments

The revenue-passenger model introduced in this paper is tested on artificial instances based on the data sets `grid` and `mandl` from the open source software library LinTim [20, 21]. The PTNs provided for each of the data sets can be used to compute network and beeline distances between any pair of stations. The distributions of the demand with respect to the network and beeline distances is shown in Figure 2. Data set `mandl` consists of 172 OD pairs that have 72 different network distances and 84 beeline distances. While data set `grid` even has 567 OD pairs, these belong only to 8 network distances and 14 beeline distances. An overview of the parameters for generating the artificial instances is given in Table 1: The demand data provided in LinTim is split into $G \in \{1, 3, 5\}$ demand groups to create the input demand data of the revenue-passenger model in four different ways (EQUAL, RANDOM, INCREASING, DECREASING). The willingness to pay for each group is generated using a flat tariff (w-FLAT) or an affine distance tariff where the distance is derived from the network (w-NETWORK) or the Euclidean distance (w-BEELINE). The parameters f and p are chosen from three options for affine distance tariffs and one option for flat tariffs. In total, this yields 252 instances per data set. The instances are solved for the revenue-passenger models F-RPM and D-RPM, determining FLAT, NETWORK distance and BEELINE distance tariffs. The solution methods are implemented in Python and the experiments are run on a machine with an Intel(R) Core(TM) i5-1335U and 32 GB of RAM.

Running Time. The running times of Algorithm 1 and Algorithm 2 are depicted in Figure 3. According to Theorem 7 and Theorem 13 the running time of Algorithm 1 is quasilinear in the total number of demand groups while the running time of Algorithm 2 is cubic. This can be observed in the running times: F-RPM can be solved in 0.08 seconds for all `grid` instances and in 0.14 seconds for all `mandl` instances, while the running time of D-RPM increases to up to 13 seconds for `grid` and to 46 seconds for `mandl`, respectively.



(a) Demand data for data set `grid`.

(b) Demand data for data set `mandl`.

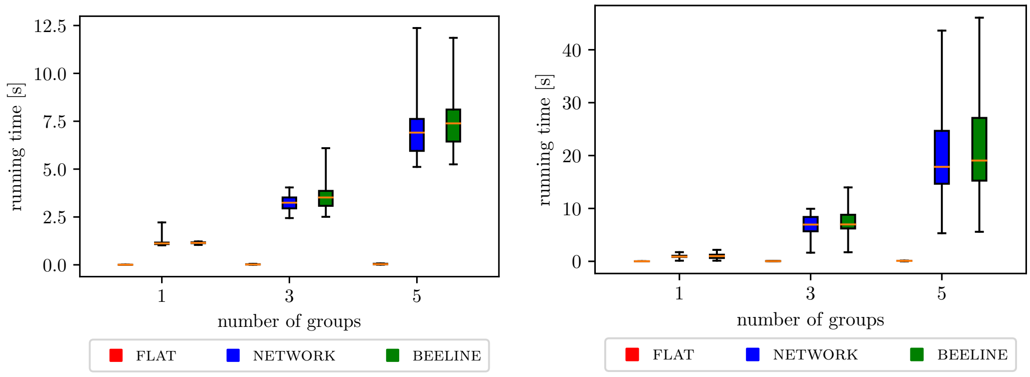
Figure 2 Demand data with respect to the different PTNs. The size of a point reflects on the demand. Above and on the right hand side of the plots, the demand with the same network or beeline distance, respectively, is aggregated.

Table 1 Parameters for generating artificial instances.

Parameter	Value	Explanation
demand groups	$G \in \{1,3,5\}$	number of groups $G_d = G$ for all OD pairs $d \in D$
demand split	EQUAL RANDOM INCREASING DECREASING	$\forall d \in D, \forall g \in [G] : t_d^g = \left\lceil \frac{t_d}{G} \right\rceil$ $\forall d \in D, \forall g \in [G] : t_d^g \in \{1, \dots, t_d\}$ random with $\sum_{g=1}^{G} t_d^g = t_d$ $\forall d \in D, \forall g \in [G-1] : t_d^g = \left\lceil \frac{t_d}{2^g} \right\rceil$ and $t_d^G = \left\lfloor \frac{t_d}{2^{G-1}} \right\rfloor$ $\forall d \in D, \forall g \in [G-1] : t_d^g = \left\lceil \frac{t_d}{2^{G+1-g}} \right\rceil$ and $t_d^1 = \left\lfloor \frac{t_d}{2^{G-1}} \right\rfloor$
willingness to pay	w-FLAT w-NETWORK w-BEELINE	tariff used to generate willingness to pay
tariff parameter	A B C	$\forall g \in [G] : f_g = g, p_g = 0.2$ $\forall g \in [G] : f_g = g, p_g = 0.6 - 0.1g$ $\forall g \in [G] : f_g = 1, p_g = 0.1g$

Figure 2 shows that the input data of `grid` is very structured and that only a few different distances occur, especially for the network distance. As suggested in Remark 14, this reflects on the running times, which is smaller for `grid` than for `mandl`, even though `grid` has roughly three times as many OD pairs as `mandl`.

These running times are orders of magnitude smaller compared to the running times of the MILP-based approach using Gurobi 10.01 [14] for solving the MILP of D-RPM. Figure 4 and Table 2 show that Algorithm 2 for D-RPM, that exploits the structure of distance tariffs, is much faster than the MILP-based approach. For NETWORK D-RPM, in 68% of the instances of `grid` with 5 demand groups, it was in some iteration not possible to even



(a) Data set `grid`.

(b) Data set `mandl`.

Figure 3 Running times in seconds for computing the complete Pareto fronts with Algorithm 1 for F-RPM and with Algorithm 2 for NETWORK and BEELINE D-RPM.

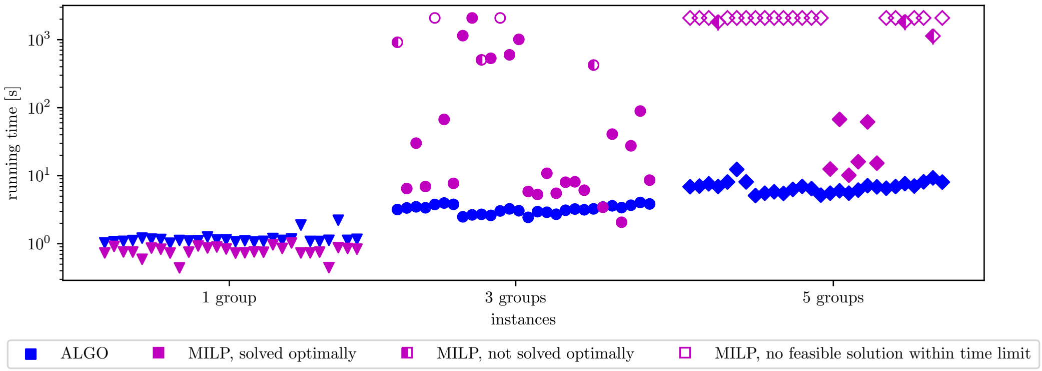


Figure 4 Running times in seconds of NETWORK D-RPM with Algorithm 2 (ALGO) and with the MILP-based method (MILP) on data set `grid` with a logarithmic scale. Each marker represents the running time for computing the Pareto front of a single instance. The time limit for solving each MILP within the ϵ-constraint method is set to 300 seconds. If a MILP could not be solved to optimality within this time limit but a feasible solution was found, then we continue with this feasible solution and label the instance as "MILP, not solved optimally". If no feasible solution is found, the procedure terminates and we label the instance as "MILP, no feasible solution within time limit" and depict it with the maximum running time in this figure.

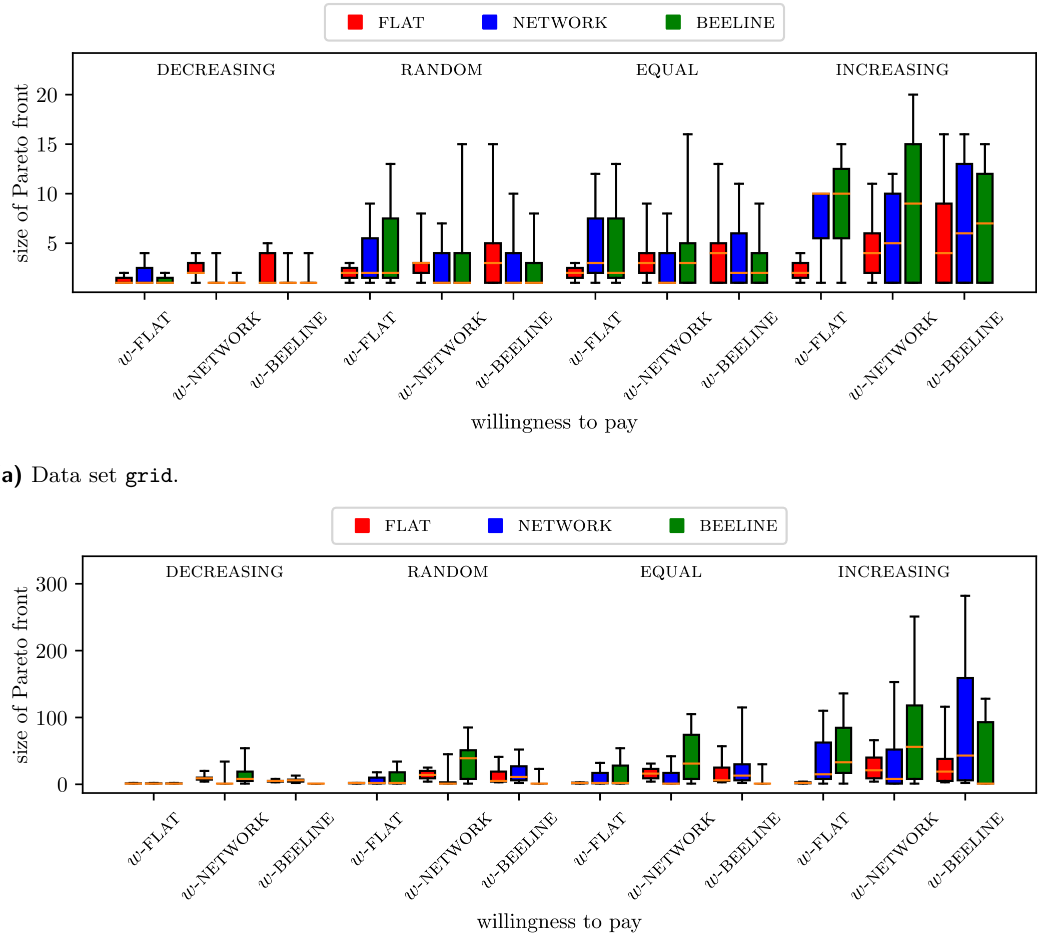
Table 2 Mean, minimum and maximum running times in seconds for solving NETWORK D-RPM on the `grid` instances with Algorithm 2 (ALGO) and with the MILP-based method (MILP). Only the instances that were solved optimally are considered.

	running time ALGO			**running time MILP**		
groups	**mean**	**min**	**max**	**mean**	**min**	**max**
1	1.19	1.02	2.21	0.78	0.44	1.02
3	3.23	2.44	4.04	248.36	2.07	2086.07
5	6.95	5.11	12.37	30.53	10.15	67.35
all	3.79	1.02	12.37	103.81	0.44	2086.07

determine a feasible solution within the time limit. With 3 demand groups this still happened to 7% of the instances. In both cases, 11% terminated with a feasible, but not necessarily optimal solution. Just in case of only 1 demand group, the MILP-based approach performs slightly better than Algorithm 2 with a mean running time of 0.78 seconds compared to 1.19 seconds. Note that strengthening the formulation of the MILP could improve the running time of the MILP-based approach.

Size of the Pareto Front. Figure 5 shows the number of points on the Pareto front for the different options for the demand splits and for the generation of the willingness to pay. We can observe two main effects:

First, a DECREASING demand split leads to a small size of the Pareto front. This is because the price increase cannot compensate for loosing large demand groups with a low willingness to pay. We see the reverse effect for INCREASING which leads to the most points on the Pareto front because loosing only small demand groups with a low willingness to pay is compensated in the revenue by the increased price.



(a) Data set `grid`.

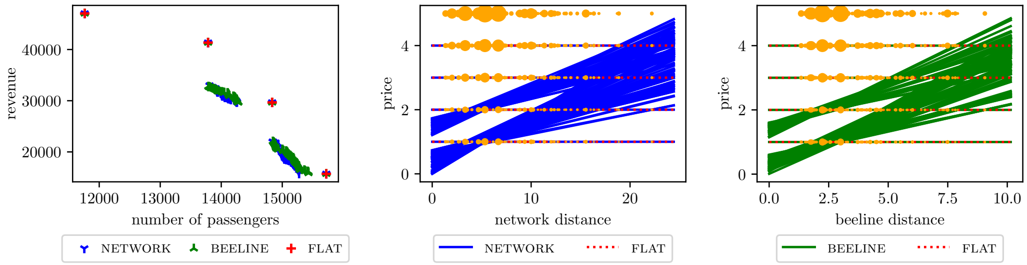
(b) Data set `mandl`.

Figure 5 Size of the Pareto front dependent on the demand split and the tariff used to generate the willingness to pay.

Second, if the fare strategy of the input tariff (for generating the willingness to pay) and the output tariff coincide, the size of the Pareto front is smaller. In this case, an output tariff might be chosen exactly as the willingness to pay of one demand group level, which is in general not possible if they differ.

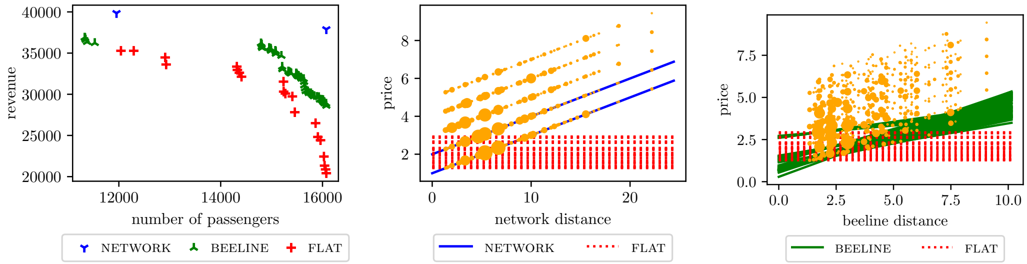
Structure of the Pareto Front, Efficient Tariffs and Input Data. Figure 6 to Figure 9 show the Pareto fronts in (a) and corresponding efficient solutions in (b) and (c) for selected parameter settings for the `mandl` instances. Additionally, (b) and (c) show the demand as points (l_d, w_d^g) weighted with the number of potential passengers t_d^g. The figures for `grid` can be found in Appendix A. In these cases, we can see well that coinciding input and output tariffs lead to a small sized Pareto front that even dominates many of the points of the other tariff types. For w-beeline, the Pareto front of beeline D-RPM dominates the Pareto front of network D-RPM, and vice versa for w-network. Particularly in Figure 6, it is visible that the distinct points on the Pareto front belong to solutions that are a flat tariff. Only in this setting with the willingness to pay being generated by w-flat, we obtain a Pareto front for F-RPM that is not dominated by both, the Pareto fronts of beeline and network D-RPM. This is however not surprising because a flat tariff is a special case of a distance tariff.

Moreover, in many cases, the efficient tariffs are located on the lower levels of the demand groups, meaning that it is not beneficial to increase the price to the highest willingness to pay. For example, Figure 6 constitutes an exception, where it is an efficient solution to choose a flat tariff with a fixed price equal to the second highest willingness to pay. However, we also see here that the highest willingness to pay does not lead to an efficient solution.



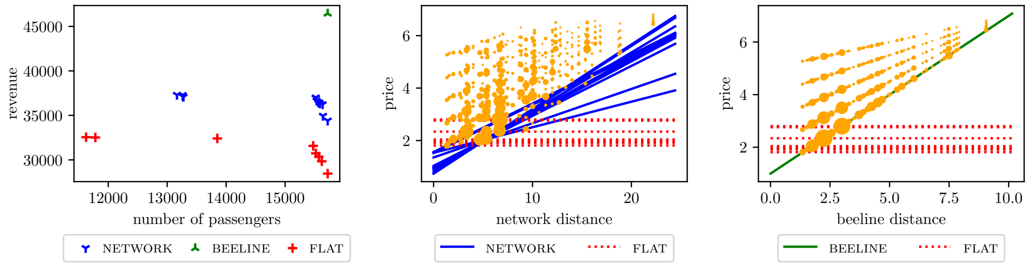
(a) Pareto fronts. **(b)** network and flat tariff. **(c)** beeline and flat tariff.

Figure 6 Instance of `mandl` with 5 demand groups and parameters increasing/w-flat/A.



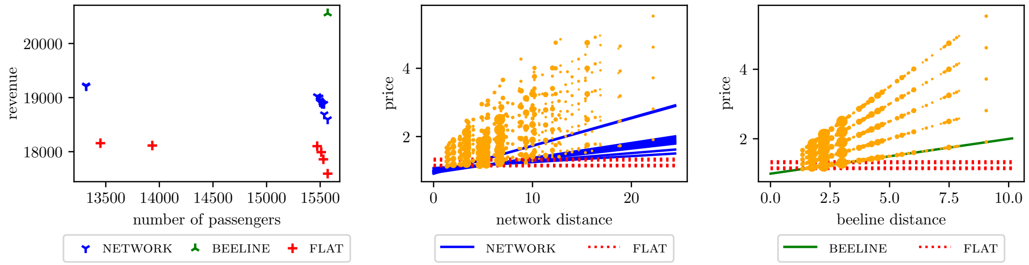
(a) Pareto fronts. **(b)** network and flat tariff. **(c)** beeline and flat tariff.

Figure 7 Instance of `mandl` with 5 demand groups and parameters random/w-network/A.



(a) Pareto fronts. **(b)** NETWORK and FLAT tariff. **(c)** BEELINE and FLAT tariff.

Figure 8 Instance of `mandl` with 5 demand groups and parameters DECREASING/w-BEELINE/B.



(a) Pareto fronts. **(b)** NETWORK and FLAT tariff. **(c)** BEELINE and FLAT tariff.

Figure 9 Instance of `mandl` with 5 demand groups and parameters EQUAL/w-BEELINE/C.

6 Discussion and Outlook

In this paper, we have introduced a bi-objective model for fare planning maximizing the revenue and the number of passengers with different demand groups per OD pair. Specialized algorithms for flat and distance tariffs showed a significant reduction in running time in computational experiments on artificial data sets from the software library LinTim.

Another common fare strategy are zones tariffs. For counting zone tariffs, for which the price depends on the number of traversed zones, the MILP-based method can be performed by applying the MILP proposed by [19] and adding the constraint restricting the number of passengers. However, because this MILP has a high running time in practice, it cannot be expected to compute the whole Pareto front, even for small instances. Future work could encompass the design of a specialized algorithm for zone tariffs.

Computational experiments show that it is worth to look into the revenue-passenger model for the specific fare strategies. Exploiting the structure of tariffs, leads to methods that allow for the computation of the complete Pareto front. This yields a wide range of information for public transport operators to choose a tariff that serves their financial requirements as well as promotes public transport with the aim to attract and increase the demand.

References

1 L. Babel and H. Kellerer. Design of tariff zones in public transportation networks: theoretical results and heuristics. *Mathematical Methods of Operations Research*, 58(3):359–374, December 2003. `doi:10.1007/s001860300311`.

2 P. Bloomfield and W. L. Steiger. *Least Absolute Deviation. Theory, Applications and Algorithms*. Progress in Probability. Birkhäuser Boston, MA, 1983.

3 R. Borndörfer, M. Karbstein, and M. E. Pfetsch. Models for fare planning in public transport. *Discrete Applied Mathematics*, 160(18):2591–2605, December 2012. doi:10.1016/j.dam.2012.02.027.

4 J.-F. Bérubé, M. Gendreau, and J.-Y. Potvin. An exact ϵ-constraint method for bi-objective combinatorial optimization problems: Application to the Traveling Salesman Problem with Profits. *European Journal of Operational Research*, 194(1):39–50, April 2009. doi:10.1016/j.ejor.2007.12.014.

5 J. Camargo Pérez, M. H. Carrillo, and J. R. Montoya-Torres. Multi-criteria approaches for urban passenger transport systems: a literature review. *Annals of Operations Research*, 226(1):69–87, March 2015. doi:10.1007/s10479-014-1681-8.

6 A. Chin, A. Lai, and J. Y. J. Chow. Nonadditive Public Transit Fare Pricing Under Congestion with Policy Lessons from a Case Study in Toronto, Ontario, Canada. *Transportation Research Record*, 2544(1):28–37, January 2016. doi:10.3141/2544-04.

7 T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 3. edition, 2009.

8 R. Cupec, R. Grbić, K. Sabo, and R. Scitovski. Three points method for searching the best least absolute deviations plane. *Applied Mathematics and Computation*, 215(3):983–994, 2009.

9 M. Czerliński and M.S. Bańka. Ticket tariffs modelling in urban and regional public transport. *Archives of Transport*, 57(1):103–117, 2021. doi:10.5604/01.3001.0014.8041.

10 M. S. Daskin, Joseph L. Schofer, and A. E. Haghani. A quadratic programming model for designing and evaluating distance-based and zone fares for urban transit. *Transportation Research Part B: Methodological*, 22(1):25–44, 1988. doi:10.1016/0191-2615(88)90032-X.

11 M. Ehrgott. *Multicriteria optimization*, volume 491. Springer Science & Business Media, 2005.

12 D. Fleishman, N. Shaw, A. Joshi, R. Freeze, and R. Oram. *Fare Policies, Structures, and Technologies*. Number 10 in TCRP Resport. Transportation Research Board, 1996.

13 D. Gattuso and G. Musolino. A Simulation Approach of Fare Integration in Regional Transit Services. In Frank Geraets, Leo Kroon, Anita Schoebel, Dorothea Wagner, and Christos D. Zaroliagis, editors, *Algorithmic Methods for Railway Optimization*, Lecture Notes in Computer Science, pages 200–218, Berlin, Heidelberg, 2007. Springer. doi:10.1007/978-3-540-74247-0_10.

14 Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2024. URL: https://www.gurobi.com.

15 H. W. Hamacher and A. Schöbel. On Fair Zone Designs in Public Transportation. In Joachim R. Daduna, Isabel Branco, and José M. Pinto Paixão, editors, *Computer-Aided Transit Scheduling*, Lecture Notes in Economics and Mathematical Systems, pages 8–22, Berlin, Heidelberg, 1995. Springer. doi:10.1007/978-3-642-57762-8_2.

16 H. W. Hamacher and A. Schöbel. Design of Zone Tariff Systems in Public Transportation. *Operations Research*, 52(6):897–908, December 2004. doi:10.1287/opre.1040.0120.

17 R. Hoshino and J. Beairsto. Optimal pricing for distance-based transit fares. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018. doi:10.1609/aaai.v32i1.11413.

18 S. Maadi and J.-D. Schmöcker. Route choice effects of changes from a zonal to a distance-based fare structure in a regional public transport network. *Public Transport*, 12(3):535–555, October 2020. doi:10.1007/s12469-020-00239-9.

19 B. Otto and N. Boysen. Zone-based tariff design in public transportation networks. *Networks*, 69(4):349–366, 2017. doi:10.1002/net.21731.

20 P. Schiewe, A. Schöbel, S. Jäger, S. Albert, C. Biedinger, C. Dahlheimer, V. Grafe, S. Roth, A. Schiewe, F. Spühler, M. Stinzendörfer, and R. Urban. LinTim - integrated optimization in public transportation. Homepage. https://www.lintim.net/.

21 P. Schiewe, A. Schöbel, S. Jäger, S. Albert, C. Biedinger, C. Dahlheimer, V. Grafe, S. Roth, A. Schiewe, F. Spühler, M. Stinzendörfer, and R. Urban. LinTim: An integrated environment for mathematical public transport optimization. Documentation for version 2024.08. Technical report, Rheinland-Pfälzische Technische Universität Kaiserslautern-Landau, 2024. URL: https://nbn-resolving.org/urn:nbn:de:hbz:386-kluedo-83557.

22 J.-D. Schmöcker, A. Fonzone, S. Moadi, and F. Belgiawan. *Determining Fare Structures: Evidence and Recommendations from a Qualitative Survey among Transport Authorities*. European Metropolitan Transport Authorities (EMTA), October 2016. `doi:10.13140/RG.2.2.17458.20162`.

23 A. Schöbel. *Locating Lines and Hyperplanes — Theory and Algorithms*. Number 25 in Applied Optimization Series. Kluwer, 1999.

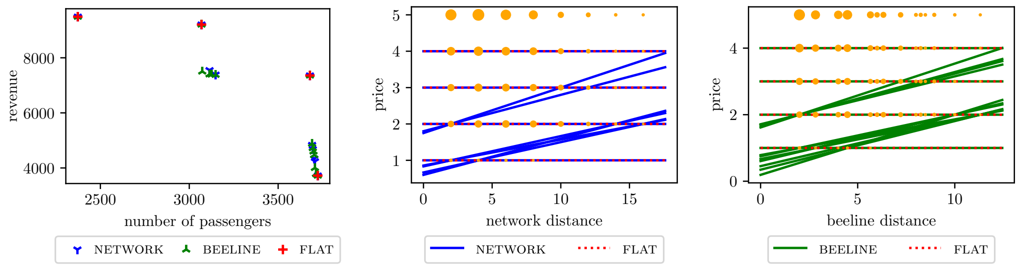
24 A. Schöbel. Locating dimensional facilities in a continuous space. In G. Laporte, S. Nickel, and F. Saldanha da Gama, editors, *Location Science*, chapter 7, pages 143–184. Springer, 2020.

25 A. Schöbel and R. Urban. The cheapest ticket problem in public transport. *Transportation Science*, 56(6):1432–1451, 2022. `doi:10.1287/trsc.2022.1138`.

26 D. Yook and K. Heaslip. Determining Appropriate Fare Levels for Distance-Based Fare Structure: Considering Users' Behaviors in a Time-Expanded Network. *Transportation Research Record*, 2415(1):127–135, January 2014. `doi:10.3141/2415-14`.

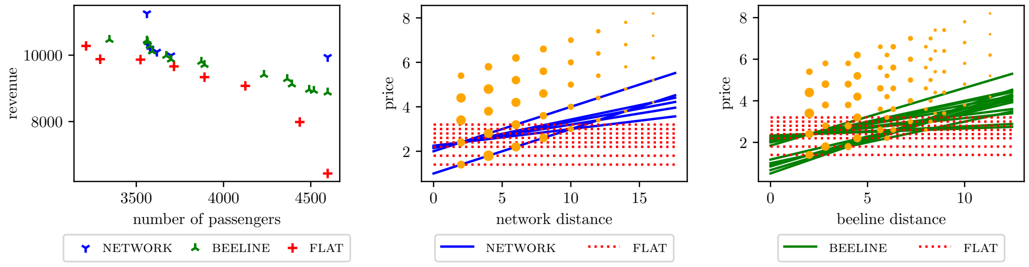
A Pareto Fronts and Efficient Solutions for Selected `grid` Instances

Figure 10 to Figure 13 show the Pareto fronts in (a) and corresponding efficient solutions in (b) and (c) for selected parameter settings for the `grid` instances. Additionally, (b) and (c) show the demand as points (l_d, w_d^g) weighted with the number of potential passengers t_d^g.



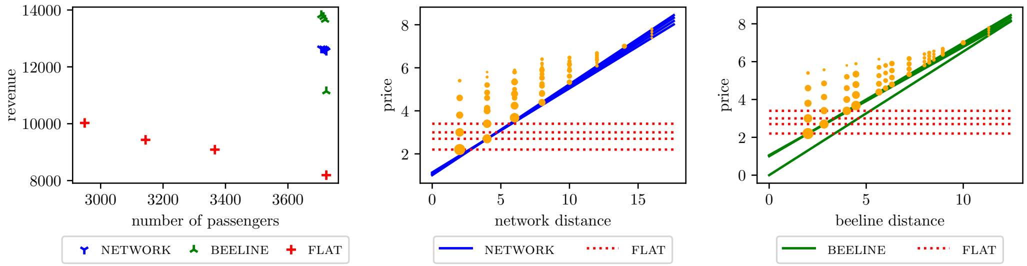
(a) Pareto fronts. **(b)** NETWORK and FLAT tariff. **(c)** BEELINE and FLAT tariff.

Figure 10 Instance of `grid` with 5 demand groups and parameters INCREASING/w-FLAT/A.



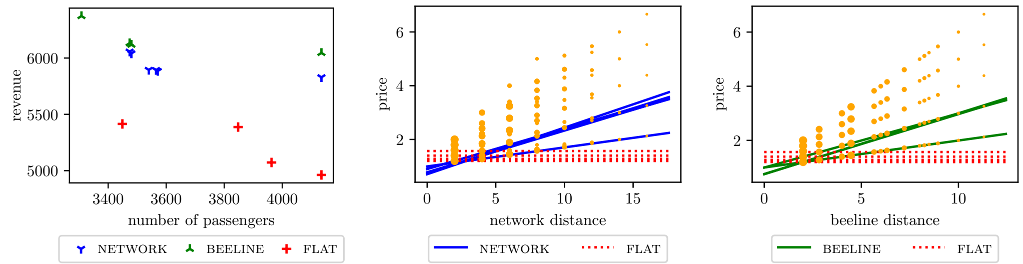
(a) Pareto fronts. **(b)** NETWORK and FLAT tariff. **(c)** BEELINE and FLAT tariff.

Figure 11 Instance of `grid` with 5 demand groups and parameters RANDOM/w-NETWORK/A.



(a) Pareto fronts. **(b)** NETWORK and FLAT tariff. **(c)** BEELINE and FLAT tariff.

Figure 12 Instance of `grid` with 5 demand groups and parameters DECREASING/w-BEELINE/B.



(a) Pareto fronts. **(b)** NETWORK and FLAT tariff. **(c)** BEELINE and FLAT tariff.

Figure 13 Instance of `grid` with 5 demand groups and parameters EQUAL/w-BEELINE/C.

Modeling Subway Networks and Passenger Flows

Antoine Thébault
Univ. Rennes, IRISA, CNRS & INRIA, Rennes, France
Alstom Transport, Saint-Ouen, France

Loïc Hélouët
Univ. Rennes, IRISA, CNRS & INRIA, Rennes, France

Kenza Saiah
Alstom Transport, Saint-Ouen, France

Abstract

Simulation of urban rail networks provides useful information to optimize traffic management strategies w.r.t. goals such as satisfaction of passenger demands, adherence to schedules or energy saving. Many network models are too precise for the analysis needs, and do not exploit concurrency. This results in an explosion in the size of models, and long simulation times. This paper presents an extension of Petri nets that handles trajectories of trains, passenger flows, and scenarios for passenger arrivals. We then define a fast event-based simulation scheme. We test our model on a real case study, the Metro of Montreal, and show that full days of train operations with passengers can be simulated in a few seconds, allowing analysis of quantitative properties.

2012 ACM Subject Classification Computing methodologies → Modeling and simulation

Keywords and phrases Subways, Passenger Flows, Modelization, Petri-Nets, Trajectory-Nets

Digital Object Identifier 10.4230/OASIcs.ATMOS.2024.16

Funding Work supported by ANRT CIFRE Grant No 2022-0444.

1 Introduction

Development of urban transport networks such as metros is a key issue in the development of cities. Beyond infrastructure, efficient traffic management algorithms are needed to plan operations in the network, and take the most appropriate decisions to optimize its performance. Traffic management can be seen as a combination of planning (one needs a priori schedules to provide a transport offer to clients, and plan composition of train fleets in metro lines), control (to take decisions online to handle delays caused by incidents, and avoid their propagation), and optimization of key performance indicators (KPIs).

Several quality criteria are usually addressed at the same time. An obvious one is passengers satisfaction, which is highly correlated with waiting times in stations. This means that to satisfy passengers demand, traffic management has to consider issues raised by crowds in stations, and by delays. Other quality criteria address running cost of a network, energy consumption, adherence to a determined schedule, etc. To answer all the needs of these complex systems, it is natural to consider automated techniques. However, models for metros are usually huge: even for a small network with a single line, a few stops and a small size fleet of metros, the size of models that can capture the dynamics of the network exceed several millions of states [3]. This exceeds the limits of most verification and optimization tools that rely on an explicit state representation. This calls for the use of efficient simulation techniques and statistical approaches to address real-size case studies.

Related Work. Several commercial tools propose realistic models to simulate metro networks. OpenTrack [20] is often used to measure the KPIs achieved under a certain traffic management policy. OpenTrack models are precise, and almost digital twins of the running systems. As a

24th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2024).
Editors: Paul C. Bouman and Spyros C. Kontogiannis; Article No. 16; pp. 16:1–16:20
OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

consequence, the simulation of a day of operation requires important computing resources. The Railsys suite [7] is another tool addressing train operations at a microscopic level. The simulation scheme of [11] starts from timetables. It considers how primary delays propagate in a schedule, and uses symbolic descriptions of train trips and reduction rules to represent states in a compact way and speed up the simulation process. However, this simulation framework does not consider passengers. Models to simulate metro with their payloads have been developed. Railnet is a simulation model to help planning shared occupation of tracks between freight and passengers transport [19]. The network model is a graph and the vehicles moves are depicted as sequences of arrival and departure dates. Railnet is used to find a solution for insertion of new freight trips that do not harm passengers trains. The model proposed in [1] is an origin-destination model for passengers interconnection to forecast passenger movements and help propose new layouts for interconnections. Another simulation method proposed in [9] evaluates passenger flows in case of service interruption. The simulation is mainly a calculus of paths duration, involving access time, waiting time, and transport duration. Many other tools exist, addressing modeling and simulation at low (microscopic) or high (macroscopic) level, with a formal background inspired from graphs, queuing theory, etc. We refer to [10, 15] for surveys of the domain.

Challenges. Most of the models proposed are either too low level and require important computing resources to simulate a network, or describe metros at a higher level, but ignore passengers. An ideal model should have a semantics close enough to the behavior of the running system, but yet allow fast techniques to perform large simulation campaigns and produce sound statistics on the performance of the system. The basic ingredients of the model must take care of the network topology, of varying composition of vehicle fleets, timetables, and passengers. Of course, an appropriate model for a metro network is a timed model (performance of traffic management is often measured w.r.t. delays or end-to-end trip durations) but also a stochastic model, as trips and dwell durations are subject to random perturbations: unexpected delays can be imposed to a vehicle by a passenger blocking a door, or by a choice of a driver to extend the dwell time in case of important crowds on platforms. As already mentioned, state space explosion is an issue for tools and models relying on explicit states representation. Further, as train movements are independent as long as the distance between the two vehicles is sufficient, concurrency models such as Petri nets are natural candidates to design metros.

Contributions. In this paper, we propose an efficient simulation scheme for metros operated with a moving block policy. Our model includes train movements, passenger flows in stations, platforms and vehicles. Important situations such as delayed trains are difficult to address with standard models such as timed automata [2] or Time Petri nets [18], which led to proposing a new timed variant of Petri nets in [13]. We extend this work, and propose *trajectory nets*, a variant of Time Petri that includes passenger flows. Roughly speaking, a trajectory net is a Petri net in which some places represent track portions and contain forecast train trajectories instead of tokens. Passenger flows in a network are modeled by assigning an integral vector indexed by destinations to trains and platforms, and by queues representing connections between lines. Last, scenarios to describe passenger arrivals in the network are modeled as sequences of Origin-Destination matrices indexed by time periods. Despite these extensions, the model still benefits from fast simulation techniques that were developed for Time Petri nets. We show the efficiency of our simulation scheme on a real case study: we evaluate the effects of train insertions on passengers waiting times during peak hours in the metro of Montreal.

2 A subway network model

A subway network is mainly a graph, and it is natural to see a day of operation as a description of how vehicles move on this graph. A metro that follows a path in the network graph travels at a given speed between two stations, i.e. the train moves for a period of time between two events: a departure and an arrival. The other events that can be considered are incidents, or decision taken by an operator. Last, vehicles have independent moves if they are distant enough in the network, but close trains may have to adapt their speed to maintain safety distances. This calls for the use of concurrent models, and Time Petri nets [18] seem tailored for this task: the flow relation of a net can represent the network, tokens can be used to represent trains, and the timing constraints attached to transitions can be used to enforce trips durations or dwell times. It was shown however that Time Petri nets are not expressive enough to model simple situations where a train has to wait for a departure order even when its dwell time has expired. A new model called waiting nets was proposed to solve this issue in [13]. Further, speed adaptation for close trains cannot be captured by tokens and clocks. In the rest of this section, we describe *trajectory nets*, a model that extends Time Petri nets to handle safety distances and varying speeds of trains. Roughly speaking, a trajectory net is a Petri net in which some places can contain diagrams representing several train trajectories from a station to the next one instead of tokens. Trajectories are discretizations of space-time diagrams, a representation of trains moves frequently used to give a feedback on realized and planned train movements in a network. In the rest of the paper, configurations of a network will be mainly collections of trajectories.

Before introducing the elements of our new model, let us recall the basics of Petri nets. A Petri net is a bipartite graph composed of a set of places, denoted P that represent resources, of a set of transitions, denoted T representing actions. Places and transitions are connected via a *flow relation*, i.e. a subset of $P \times T \cup T \times P$. The set ${}^\bullet(t)$ denotes the resources (set of places) required for transition t to occur, and the set $(t)^\bullet$ represent the resources produced by execution of transition t. Petri nets are often used to model systems with a lot of concurrency and analyze their behaviors. The state of a Petri net is called a *marking*, and is a map that associates a number of *tokens* to each place. The state of a Petri net evolves by firing actions, i.e. consuming one token in each place representing a resource required by a transition t and producing one token in each place in the postset of t. Behaviors of Petri nets can be infinite, and the state state space reachable by a Petri net from an initial marking can also be infinite. Despite this expressive power, many properties such as reachability are decidable for Petri nets [17]. Figure 1 shows an example of Petri nets with places $P = \{p_1, p_2, p_3\}$ and transitions $T = \{t_1, t_2\}$. Repeating sequence of transitions $t_1.t_2$ allows to fill place p_3 with any number of tokens. Due to their graphical nature that can easily simulate networks, to their clear semantics, and to their decidability, Petri nets or their extensions are often used to model railway systems.

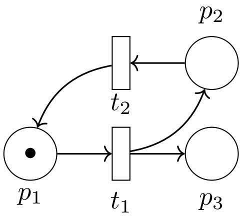


Figure 1 An example Petri net.

▶ **Definition 1.** *A* segment *is a pair of points $\langle(x,y)(x',y')\rangle$ where x,x' are real values that represent time and y,y' are real values that represent distances. A* trajectory *is a pair (tr,b), where b is a boolean indicating if the trajectory is blocked, and $tr=\langle(x_0,y_0)(x_1,y_1)\rangle\cdots\langle(x_{k-1},y_{k-1})(x_k,y_k)\rangle$ is a finite sequence of consecutive segments such that $x_0=0$, and $\forall i, x_i > x_{i-1}$ and $y_i \leq y_{i-1}$. A trajectory is* complete *if $y_k=0$.*

In a point (x,y) of a segment, x represents a duration, and y a remaining distance to a destination. So, trajectories define how distance to arrival decreases over time. For instance a trajectory $tr = \langle(0,200)(10,100)\rangle.\langle(10,100)(30,0)\rangle$ describes a train at distance 200 m from its goal, that arrives 30 seconds later. It travels at $10m/s$ during 10 seconds, and then slows down at $5m/s$. A train approaching a close predecessor will have to adapt its trajectory to preserve headways, and elapsing δ time units will simply consist in shifting trajectory segments by δ units to the left. For a trajectory $tr = \langle(x_0,y_0)(x_1,y_1)\rangle\cdots\langle(x_{k-1},y_{k-1})(x_k,y_k)\rangle$, and a date $d \leq x_k$ we will denote by $tr(d)$ the coordinate of a train at date d. For $x_i \leq d \leq x_{i+1}$ we have $tr(d) = y_i + (d-x_i).\frac{(y_{i+1}-y_i)}{(x_{i+1}-x_i)}$. We will say that two trajectories tr, tr' *respect a safety headway* h if and only if $\forall d, |tr(d)-tr'(d)| \geq h$, and that tr' is above tr if $\forall d, tr'(d)-tr(d) \geq 0$. Trajectories are defined for track portions of bounded length. For a length H we assume that for every trajectory, $y_0 \leq H$. So, we can have at most H/h trajectories respecting a safety headway h in a track segment of length H.

Trajectory nets defined hereafter will use *sequence of trajectories* of the form $TS = (tr_1,b_1)\cdots(tr_k,b_k)$ to represent trips for several trains in a given space of length H. We forbid ill-formed sequences of trajectories where trains overtake or violate the safety headway.

▶ **Definition 2.** *A sequence of trajectories $TS = (tr_1,b_1)\cdots(tr_k,b_k)$ is* consistent *(w.r.t. headway h) iff every pair of trajectories in TS respects safety headway h, and $b_i = true$ implies that $b_j = true$ for every $j < i$.*

For a sequence of trajectories $TS = (tr_1,b_1)\cdots(tr_k,b_k)$, we can denote by x_j^k (resp. y_j^k) the j^{th} value of x (resp. y) in trajectory k. With this notation, $y_0^k = tr_k(0)$ is the current distance to next stop of train k in sequence TS. Let $CTS(H,h)$ denote the set of consistent sequences of trajectories (w.r.t. headway h) in a space of length H. Given a consistent sequence of trajectories $TS = (tr_1,b_1)\cdots(tr_k,b_k)$ such that $tr_k(0) \leq H-h$, we can sample an additional trajectory tr_{k+1} such that $TS.(tr_{k+1}, false)$ is consistent and $y_0^{k+1} = H$. Let $I = [\alpha,\beta]$ be a time interval depicting a possible trip duration (these values depend on train speeds allowed by the network, the rolling stock specifications, and the policies given by operators), $d \in I$ be a rational value, and $\widehat{tr_{k+1}} = \langle(0,H)(d,0)\rangle$. Trajectory $\widehat{tr_{k+1}}$ depicts a space time diagram for a train traveling at average speed $\frac{H}{d}$ when no other train is on the track. Now, to preserve consistency of a place contents one cannot always add directly $\widehat{tr_{k+1}}$, to an existing sequence TS : the wished trajectory may have to be adapted (this amounts to reducing the speed of a train) to stay at distance h from preceding trains. That is, one has to compute a trajectory $tr_{k+1} =\uparrow(\widehat{tr_{k+1}}, TS, h)$ such that, for every time value t, $\uparrow(\widehat{tr_{k+1}}, TS, h)(t) = \max(\widehat{tr_{k+1}}(t), tr_k(t)+h)$.

Computing $\uparrow(\widehat{tr_{k+1}}, TS, h)$ is a simple geometrical calculus, defined formally in Appendix A. One can notice that by construction, $tr_{k+1} =\uparrow(\widehat{tr_{k+1}}, TS, h)$ is the only trajectory for a train running at speed H/d whenever possible such that $TS.(tr_{k+1}, false)$ is consistent. For a given sequence TS and time interval $I = [\alpha,\beta]$, we denote by $\text{SAMPLE}(TS,H,h,I) = \{\uparrow(\widehat{tr_{|TS|+1}}, TS, h) \mid \exists d \in I \wedge \widehat{tr_{|TS|+1}} = \langle(0,H)(d,0)\rangle\}$ the set of additional trajectories depicting trains with trips of length H, that can be added consistently to TS while respecting headway h with an initial speed in interval $[\frac{H}{\beta}, \frac{H}{\alpha}]$. This way of sampling trajectories allows the introduction of random perturbations w.r.t. a chosen speed

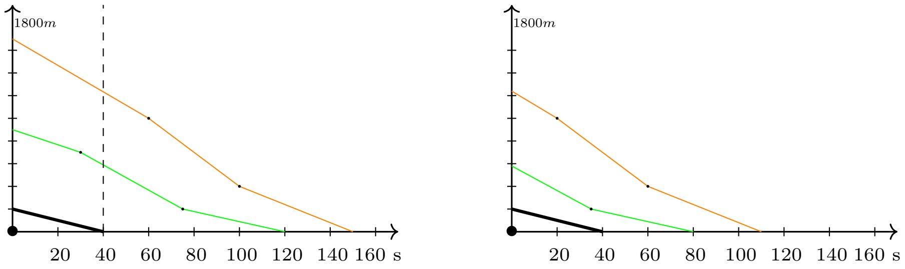


Figure 2 A sequence of trajectories (left) and its left shift by $\delta = 40s$ (right).

profile with an appropriate probability distribution on I. The sampled trajectories can be simple segments respecting headways (such as the black trajectory in Figure 2), or sequences of consecutive segments at distance $\geq h$ from preceding train (such as the orange and green trajectories in Figure 2).

As one can see on Figure 2, trajectories have an intuitive graphical representation, showing, after x time units the distance from a train to the next station. A trajectory that is blocked represents a train that cannot move because it does not have a sufficient headway ahead. Let us now give a graphical intuition of how trajectories evolve when time progress. We will call the *left shift* of a sequence of trajectories by a real value δ the sequence of trajectories obtained by letting trajectories of blocked trains unchanged, and translating all other trajectories by a value $-\delta$. Figure 2 shows an example of left shift. In the original configurations, two trajectories are blocked : a train at distance 0 from arrival (represented by a dot) and its successor, that has to stop to preserve a headway, represented by a thick black line. The next two trains are at a sufficient distance to continue their planned trip during 20 seconds. The left shift of the initial positions of trains by 20 seconds simply consists in a translation of 20 units to the left of unblocked trajectories (erasing the parts of shifted trajectories with negative absiscaes). As for trajectory creation, left shift of a trajectory tr by δ time units is a simple geometrical operation, denoted by $LS(tr, \delta)$, that we define formally in Appendix B. We now have all elements to define our simulation model for metro networks.

▶ **Definition 3.** *A* trajectory net *is a tuple $\mathcal{N} = (P, T, F, H, I)$ where $P = P_C \cup P_T$ is a set of places partitioned into control places P_C and trajectory places P_T. T is a set of transitions, and $F \subseteq P \times T \cup T \times P$. Maps $H : P_T \rightarrow \mathbb{Q}$ associates a length, and $I : P_T \rightarrow \mathbb{Q}^2$ a time interval with each trajectory place of P_T.*

As usual, we denote by ${}^\bullet(t) = \{p \in P \mid (p, t) \in F\}$ the preset of a transition $t \in T$ and by $(t)^\bullet = \{p \in P \mid (t, p) \in F\}$ its postset. A *marking* M of P_C is a map that associates an integral number of tokens with every control place in P_C. We say that M enables a transition $t \in T$ iff, for every place $p \in P_C \cap {}^\bullet(t)$, $M(p) > 0$. The effect of a firing of a transition t on a marking M that enables it is to produce a new marking M' obtained by removing a token from each place in ${}^\bullet(t) \cap P_C$ and then adding a token in each place of $(t)^\bullet \cap P_C$. With a slight abuse of notation, we will write $M' = M - {}^\bullet(t) + (t)^\bullet$. Let $H_{max} = \max_{p \in P_T} H(p)$, and assume a fixed headway h for the whole network. A *trajectory marking* (or TMarking for short) is a map $\mu : P_T \rightarrow CTS(H_{max}, h)$ that associates a sequence of trajectories with each place of P_T. We will say that μ enables t if, for every place $p \in {}^\bullet(t) \cap P_T$, $\mu(p)$ is not empty and contains a trajectory of the form $(tr, true)$, with $tr = \langle(0,0)(0,0)\rangle$ depicting a train arrived at the end of its trip. A *configuration* of a net $\mathcal{N}$ is a pair $C = (M, \mu)$ where M is a marking, and μ a TMarking. Figure 3 is a simple trajectory net with two control places p_3, p_4, two trajectory places p_1, p_2 and a transition t. The TMarking of place p_1 is a sequence of two trajectories, where the green trajectory was adapted to preserve safety headways.

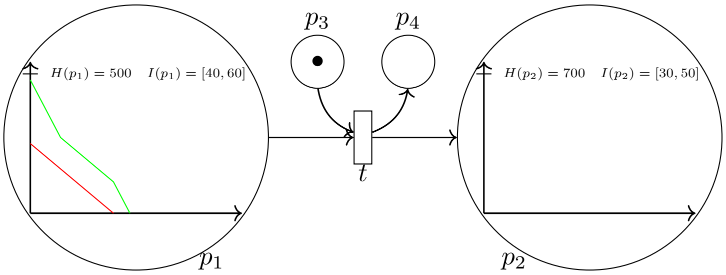

p_3
p_4
$H(p_1) = 500$ $I(p_1) = [40, 60]$
$H(p_2) = 700$ $I(p_2) = [30, 50]$
t
p_1
p_2

Figure 3 Basic elements of a trajectory net: places, transitions, trajectories.

The semantics of trajectory nets is given in terms of discrete and timed moves from a configuration to the next one. Discrete moves depict arrivals and departures of trains, trajectory changes imposed to preserve safety headways, and timed moves depict how trajectories evolve when time elapses. One can simulate a train waiting in a station S with a trajectory place p_S representing the station, such that $H(p_S) < h$. In this setting, a TMarking of place p_S can contain at most one train, which waiting duration lays in $I(p_S)$. Adding a trajectory in place p_S models an arrival in station, removing a trajectory a departure.

Given a non-empty TMarking $\mu(p) = (tr_0, b_0) \cdots (tr_k, b_k)$ one can easily find the remaining time $\delta_{arr}(p)$ before arrival of a train depicted by trajectories in $\mu(p)$ if tr_0 is not blocked, or $\delta_{block}(p)$ before a trajectory in $\mu(p)$ has to be blocked to preserve consistency of $\mu(p)$. For a sequence $TS = (tr_0, b_0) \cdots (tr_k, b_k)$, $\mathrm{UNBLOCK}(TS) = (tr_0, false) \cdots (tr_k, false)$ is the sequence obtained by unblocking all trajectories in TS.

Consider semantics rule $R1$ below, depicting an arrival or a departure. Roughly speaking, the rule consists in "consuming" a complete trajectory representing a train arrived in station, creating new ones in the places[1] of $(t)^\bullet$, and unblocking trajectories of trains that were blocked until this arrival. The second semantics rule $R2$ below describes how trains can get blocked when there is not enough space ahead to move.

$$(R1)\qquad \frac{\begin{array}{c} \text{M enables } t,\ \mu \text{ enables } t,\ M' = M - {}^\bullet(t) + (t)^\bullet \\ \forall p \in P_T \cap {}^\bullet(t), \mu(p) = (\langle(0,0)(0,0)\rangle, true).W \wedge \mu'(p) = \mathrm{UNBLOCK}(W) \\ \forall p' \in P_T \cap (t)^\bullet, \mu'(p') = \mu(p') \cdot (tr, false),\ \text{where } tr \in \mathrm{SAMPLE}(\mu(p'), H, h, I(p)) \end{array}}{C = (M, \mu) \xrightarrow{t} C' = (M', \mu')}$$

$$(R2)\qquad \frac{\begin{array}{c} \mu(p) = (tr_1, b_1) \cdot (tr_2, b_2) \cdots (tr_i, b_i) \cdots (tr_k, b_k) \\ b_i = false \wedge \big((b_{i-1} = true \wedge tr_i(0) - tr_{i-1}(0) = h) \vee tr_i = \langle(0,0)(0,0)\rangle\big) \\ \mu'(p) = \{(tr_1, b_1), (tr_2, b_2) \cdots (tr_i, b_i' = true) \cdots (tr_k, b_k)\} \end{array}}{C = (M, \mu) \xrightarrow{blockp,i} C' = (M, \mu')}$$

Timed moves just let time elapse. We adopt an urgent semantics: a timed move of duration δ is allowed from configuration C if and only if no discrete move is allowed in C. Unsurprisingly, letting time pass is modeled by a left shift of TMarkings. However, blocked trajectories represent trains that cannot move without violating a safety headway. Hence, the left shift of a TMarking $\mu(p) = \{(tr_1, b_1) \cdots (tr_k, b_k)\}$ by a duration δ is a new TMarking $\{(tr_1', b_1) \cdots (tr_k', b_k)\}$ where $tr_i' = LS(tr_i, \delta)$ if $b_i = false$ and $tr_i = tr_i'$ otherwise.

[1] To represent metro networks, one only needs $|(t)^\bullet| = 1$ but this restriction is not needed in the semantics.

$$(R3)\quad \frac{\begin{array}{c}\delta \in \mathbb{R}^{>0}\\ \forall t \in T, M \text{ does not enable } t \vee \mu \text{ does not enable } t\\ \forall p \in P_T \text{ such that } \mu(p) \text{ is defined } \delta \leq min(\delta_{arr}(p), \delta_{block}(p))\\ \forall p \in P_T, \mu'(p) = LS(\mu(p), \delta)\end{array}}{C = (M, \mu) \xrightarrow{\delta} C' = (M', \mu')}$$

With these semantic rules, we can simulate the behavior of a metro network represented by a trajectory net. Arrivals and departure are symbolized by transitions firings, and result in the sampling of a new trajectory that is consistent with the possible speeds of trains and with the TMarking of the new track entered. An interesting feature of Rule 3 is that one can directly elapse time for a duration $\delta = \min_{p \in P_T} \min(\delta_{arr}(p), \delta_{block}(p))$, i.e. progress time up to the date of the next discrete event : arrival/departure in station, or blocking of a trajectory. With this approach, one does not need to sample a discrete clock and compute what happened at each time step. This approach is called *event-based simulation*, and considerably speeds up simulation of metro networks. One can also see from these rules the advantages of using Petri nets variants: in each discrete rule, the effect of an event changes the contents of a single place, or the preset/postset of a single transition.

We can now define runs of a trajectory net. A *run* is a sequence $\rho = C_0 \xrightarrow{\delta_0} C_1 \xrightarrow{e_0} C_1 \cdots$ where each C_i is a configuration, $C_i \xrightarrow{\delta_i} C_{i+1}$ is a legal time move, and $C_i \xrightarrow{e_i} C_{i+1}$ is a legal discrete move (e_i is either a transition firing, or the blocking of a trajectory in a place). Note that the behavior of a net starting from a configuration C_0 is not deterministic, as transitions firings sample random durations for newly created trajectories.

Despite their apparent simplicity, trajectory nets allow for the design of complex topologies of metro networks involving forks, joins, reversal or garage zones for trains, ... Using control places filled at the appropriate moment, one can guide a train reaching a fork to enter the next track segment on its trip. We refer interested reader to Appendix F for examples.

3 Passenger flows

The model of Section 2 does not consider passengers, but only possible moves of trains. In this section, we model passenger flows as a side quantitative information evolving with runs of a trajectory net, i.e. as quantities representing crowds, that are updated at each timed and discrete move. First of all, passenger flows are not constant over time, and depend on particular scenarios. In most cities, during a working day, one can observe peak hours during which commuters move from their home to their workplace, and the way back in the evening. Scenarios for passenger flows during weekends differ w.r.t. dates and duration of peaks, number of moving passengers or destinations. To control efficiently a transport network, i.e. provide the expected transport offer but nevertheless use the right amount of resources, operators use different policies for each scenario. In this section, we first explain how scenarios for passenger flows are usually represented in metro networks (using Origin-Destination matrices), how connections between parts of a network (corridors or access to quays) are specified. We use these representations to decorate in a consistent way runs of our metro model with quantities depicting passengers that are moving towards a platform, waiting on a platform, or traveling in a train.

Roughly speaking, passenger behaviors (arrivals and final destination) are encoded as sequences of origin/destination matrices, connections as queues, and train payloads as vectors containing the number of passengers alighting at each destination in the network. To avoid overloading our model, we consider that all passengers move at identical constant speed from a point to the next one. This modeling choice may result in slight modification in the duration of transit times. Indeed all passengers do not move at the same speed, corridors may contain bottlenecks, and paths from a platform to another one on a different line may include stairs, etc. However, there is no clear consensus on how passenger flows should be modeled. It seems that for corridors with bottlenecks, the throughput depends mainly on the width of the bottleneck [21]. We will hence model passengers moving in a station by queues of integers. In a similar way, there is no canonical model to represent passengers entering or leaving a trains, and the time needed to load and unload may depend on characteristics of the rolling stock (e.g. the number of doors, their width and their position) [16]. Here again, we will adopt a simple model, and consider the exchanges that occur depending on the dwell time, passengers on platform, and available space in trains.

► **Definition 4.** *Let $\mathcal{S} = \{S_1, \cdots S_m\}$ be a list of stations in a metro network, and let us fix a duration Δ. An* Origin Destination Matrix *(ODM for short) is an integer matrix OD indexed by $\mathcal{S}$. Each entry $OD[s_i, s_j]$ represents a number of passengers willing to travel from station s_i to station s_j in Δ time units. A* scenario *is a sequence $OD_1, OD_2, ..., OD_n$ of origin destination matrices. The* duration *of a scenario is $n \cdot \Delta$*

An ODM depicts the number of passengers traveling from one origin station (represented by a row in the matrix), to a destination station (represented by a column) during a duration of Δ time units. The origin and destination stations need not be located on the same subway line. This means that passengers may have to use interconnection at stations shared by several lines, and move from one platform to another one to continue their journey. Hence, the path followed by passengers will have an impact on passenger flows in the transit areas at connection points. A metro network can be seen as a graph depicting connections from a station to the next one along each line and interconnections between two lines. We assume that this graph is consistent with the flow relation of the trajectory Petri net modeling our network. We also assume that passengers always follow the shortest path (in terms of distance or in terms of number of stations) in this graph when traveling from an origin station S_o to a destination station S_d. With this assumption one can compute the shortest path for every pair (S_o, S_d) of origin/ destination, using for instance the well known Floyd-Warshall algorithm (see for instance [5]). For completeness, we provide this algorithm in Appendix D. Let us assume that we have obtained a matrix M_D giving the shortest distance between any pair of connected stations. Then, we can compute a matrix M_{NS} such that $M_{NS}(S_i, S_j) = k$ if station S_k is the next station to visit on an optimal path going from S_i to S_j. When one knows the final destination of passengers, this matrix M_{NS} allows to decide, when a train stops at a station, how many passengers leave the train, and how many passengers continue their journey to the next station on the same line. The calculus of M_{NS} can be done with a simple extension of the Floyd-Warshall algorithm, given in Appendix E. Notice that this way of modeling passengers choices is a way to keep the model simple. We could make passenger choices more complex, e.g. allow choices of alternative routes to a destination, or choice of a particular line to go from an origin to a destination. Note that line information is not needed when lines do not share common track portions (as in the metro of Montreal).

Each ODM is used to depict passenger arrivals and destinations in the network for a duration Δ. Scenarios allow changes in the arrival dates at each station, at each period of the day. Let $d \leq \Delta$. Assuming that an ODM OD represents faithfully passenger trips, the number of passengers entered in the network at station S_i willing to go to station S_j is $\lfloor OD[i,j] \cdot (d/\Delta) \rfloor$. Remembering the time t elapsed since the beginning of a simulation, one can compute how many passengers have entered the network and their destination at time $t + \delta$. Note that this calculus may involve contents of several OD matrices.

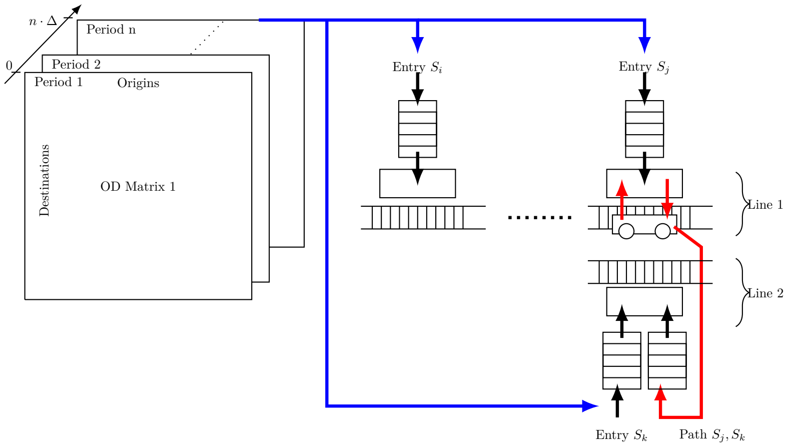


Figure 4 Integration of scenarios and passenger flows in the model.

At each instant, one wants to remember the number of passengers for a destination in a train, on platforms, in corridors connecting one entry to a platform, or on platform serving a line to another platform serving a different line in the same station. We define a *payload* as a vector of integers indexed by station, that is vectors of the form $PL = [v_1, \cdots, v_n]$. The interpretation is the following: if PL describes the passengers of a train, then $PL[i]$ indicates the number of passengers willing to reach station s_i in that train. Obviously, when a train alights at station S_i, then at least $PL[i]$ passengers leave the train. Further, if S_i is the last station on the same line on a shortest path to station S_k, then $PL[k]$ passengers leave the train too, and move towards a corridor connecting the reached platform S_i to the platform $MN[S_i, S_k]$, that is the next step on the journey to S_k. We assume that passengers do not make mistakes, i.e. they all follow the shortest path to their destination. Payloads are also used to represent awaiting passengers on a platform: $PL[i]$ is then the number of passengers that did not yet board a metro and are waiting for a train to go to station S_i.

Let us now add corridors to the model. Corridors connect entries of the network to platforms, and bridge lines to allow journeys involving several metro lines. As for trains and platforms, one has to remember the number of passengers moving in a corridor and their final destination. However, passenger moves in corridors take time, and cannot be represented with a payload. We use FIFO queues to represent passenger flows.

▶ **Definition 5.** *A* passenger queue *is an tuple $Q = (l_Q, sp_Q)$ where l_Q is a length (in meters), sp_Q is an average speed. A state of a passenger queue Q is a vector of payloads M_Q indexed by $1..l$. We will denote by $M_Q[k]$ the passengers at distance k from the beginning of the queue, and $M_Q[k][j] = n$ means that at distance k, n passengers are willing to go to station S_j.*

Each entry $M_Q[k]$ in a passenger queue represents a section of 1 meter on a path from an origin (a platform or a station entry) to a destination (a platform). We take some simplifying assumptions : the speed of passengers is constant in the whole space represented by Q and is the same for all passengers. Passengers do not interact. With these assumptions, we can simply remember with $M_Q[k][j]$ how many passengers willing to go to station S_j are at distance k from the next platform. So, using passenger queues, one need not assign a behavior to every passenger in the network (which would be too costly to simulate). More complex models for crowd behaviors have been proposed [12, 16], but we are mainly interested in durations of transfers and in destinations of passengers, not in individual movements. Further, modeling flows with queues allow for an easy calculus of the evolution of crowds. Let M_Q be the current state of a queue $Q = (l, sp)$ from an entry at station S_i to a platform, d be the current date. Assume that the played scenario is $OD_1, OD_2, ..., OD_n$. We can easily compute how the state of Q evolves within δ time units. Passengers in M_Q at distance $x \leq \delta \cdot sp$ have arrived, so the corresponding sum of payloads from 1 to x is added to the payload of the platforms. Passengers in M_Q at distance $y > x + 1$ simply progressed in the queue, i.e. $M_Q[y - \lfloor \delta \cdot v \rfloor] = M_Q[y]$. Last, assuming that d and $d + \delta$ belong to the same period, i.e. arrivals are given by a single ODM OD, we have $OD[i][j] \cdot \frac{\delta}{\Delta}$ new passengers arriving in Q willing to go to station S_j, and spread uniformly on entries $\delta \cdot sp \ldots l$ of the queue. In practice, computing the actualized contents of queues within δ time units may involve several OD matrices, and one has to pay attention to rounding to avoid loosing passengers. To avoid heavy notations, we leave the complete definition of passenger moves to a more formal definition provided in Appendix G, and we will simply write $M'_Q = \mathit{Update}(d, \delta, M_Q)$ to denote that M'_Q is the state of queue Q after δ time units have elapsed since date d. Similar rules apply for queues representing connections between platforms. In a similar way, we can represent how the payload of a train evolves when time elapses. When a train arrives at station S_i, passengers alighting at S_i leave the train, and either leave the system, or move to the queue representing the connections to another platform. Then, passengers on platform board the train up to the maximal capacity CT_{max} of the vehicle. We refer to the model of [12] to compute the time needed by n passengers to leave a train. We set $t_{alighting} = \lceil n \cdot D_{width} \times D_{nb} \rceil \times t_a$ where D_{width} is the width of a door in meter, D_{nb} is the number of doors of the vehicle and t_a is the time for one passenger to alight, estimated in seconds. Note that a train cannot leave if its passengers are still alighting, so when $t_{alighting}$ is greater than the dwell time planned, the train is delayed. When all passengers alighting at s_i have left the train, and the payload of the train is TP then the free space can be used to let $n \leq CT_{max} - \sum TP_k$ passengers enter the train. Boarding is similar to alighting time, i.e. we have $t_{boarding} = \lceil n \cdot D_{width} \times D_{nb} \rceil \times t_b$, where t_b is the average time needed by a passenger to board a train. Here, if $t_{boarding}$ is larger than the remaining dwell time of a train, then our model assumes that doors close, and that some passengers fail to board.

As the remaining space, or the dwell time of trains might not allow all passengers to board, some of them will stay on the platform. Failures to board are an important performance indicator, as this is one of the quantitative aspects that characterizes users' satisfaction. As for queues, we will denote by $TP'_i = \mathit{Update}(d, \delta, TP_i)$ the change of payload in train T_i within δ time units, and $P'_i = \mathit{Update}(d, \delta, P_i)$ the change in platform i's payload. Figure 4 gives an illustration of passenger flows between platforms, corridors, entries and trains.

Let us now reconnect trajectory nets, passenger queues and scenarios. The global state of a simulation is represented by a tuple $\big(C, d, (P_i)_{i\in 1..K_P}, (TP_i)_{i\in 1..K_T}, (M_{Q_i})_{i\in 1..K_Q}\big)$ where C is a configuration of the trajectory net depicting the physical network, d is a date, $(P_i)_{i\in 1..K_P}$ represents platforms crowds, $(TP_i)_{i\in 1..K_T}$ represents train payloads, and $(M_{Q_i})_{i\in 1..K_Q}$ are the queues contents. Let t be an arrival of a train in station s_i after a delay δ. The new state of the system is $\big(C', d + \delta, (P'_i)_{i\in 1..K_P}, (TP'_i)_{i\in 1..K_T}, (M'_{Q_i})_{i\in 1..K_Q}\big)$, where C' is the configuration reached by the trajectory net after elapsing δ time units and firing t, i.e., such that $C \xrightarrow{\delta} C'' \xrightarrow{t} C'$, and $P'_i = \mathit{Update}(d, \delta, P_i)$, $TP'_i = \mathit{Update}(d, \delta, TP_i)$, $M'_{Q_i} = \mathit{Update}(d, \delta, M'_{Q_i})$. Similar rule applies when blocking operations occur. At each use of an operational rule for a train movement, at most one train has its payload updated, but the contents of all queues and platforms change. So, one has to perform a number of updates in $O\left((K_p + K_Q \cdot L_{\max} + 1) \cdot S\right)$, where $L_{\max}$ is the maximal length of a queue. Experimental results in the next section show that this simulation model is efficient, and allows for fast simulation of a complete day of metro operation.

4 A case study: The metro of Montreal

We consider a real-size case study, namely the Metro network of Montreal (see [6] and Appendix H for a complete map). The network is composed of three main lines (orange, green, blue) plus a short yellow side line. The orange line is 31km long, it has 42 trains and 31 stations, which corresponds to 128 places and transitions in our net model. The green line is 22km long, it has 34 trains, which corresponds to 108 places and transitions in our net model. The blue line is 9km long, with 32 trains, which corresponds to 48 places and transitions. In addition to the trajectory net defining the physical behavior of trains, queues have been created for each platform, and set a limit for the capacity of platforms. We choose a default queue distance of 10m for corridors from an entry to a platform, a platform capacity of 1000 awaiting passengers, and a train capacity of 1000 passengers. We represent the 4 interconnections in the network with 8 queues (one per direction) of 50m.

The control part of the trajectory is extracted from a real GTFS file [8] (a standard proposed by Google in 2006 for timetable description). This public data depicting one day of operation on Montreal's network is available on the STM site [22]. From this timetable, we generate a list of events associating a date, a vehicle id and a transition to fire. This timetable allows to feed the control part of our model to trigger train departures. If needed, delays can be inserted to simulate small perturbations (e.g. doors blocking) or a more serious anomalies causing a technical interruption of traffic.

We simulated the trajectory net depicting Montreal's network with the MOCHY tool [14], an Open-GL software developed for the simulation of transport networks described with variants of Petri nets. The experimentation was conducted with a weekday scenario repeating the same OD matrix representing one hour of passenger flows. In this matrix, for every entry $OD[i,j]$, we generated randomly a high number of passengers entering the network at station S_i and willing to go to destination S_j. This number of passengers per hour was generated with a uniform law to sample a value ranging from 0 to 100. Multiplying an average number of 50 passengers per hour by the number of stations in the network rapidly leads to large flows of incoming passengers. With this scenario, we can model a peak period: in all stations, a large number of passengers arrive and travel from their home place to their work. In weekdays scenarios, the payload of all trains increase up to their full capacity if the operators do not inject new trains in the network. The objective of this simulation was to consider the effects of train insertion during peak hours, to verify that this policy influences train payloads. Overall, simulating a full day of operation for the three main lines with 110 trains and their passenger flows takes between 1 and 2 minutes on a standard laptop (Intel core i7). Our first simulation results are represented in Figures 5 and 6. Figure 5 is a space-time diagram drawn from our simulation logs. It represents train moves between station Montmorency and station Cote-Vertu on the Orange line from 5:00 to 8:20. The horizontal axis represents time. The vertical axis represents localization of trains, labeled by station names plus a garage, and ordered according to their position on the line. Each colored line represents a train. This simulation follows exactly the planned timetable, i.e, events have been realized exactly at their planned dates. The timetable was designed to guarantee regularity of train departures at each station, and to increase the frequency of trains to absorb passenger peaks. On the space time diagram, one can observe regularity of service: train trajectories are parallel lines. When a train is inserted, trajectories adapt and get closer, as expected.

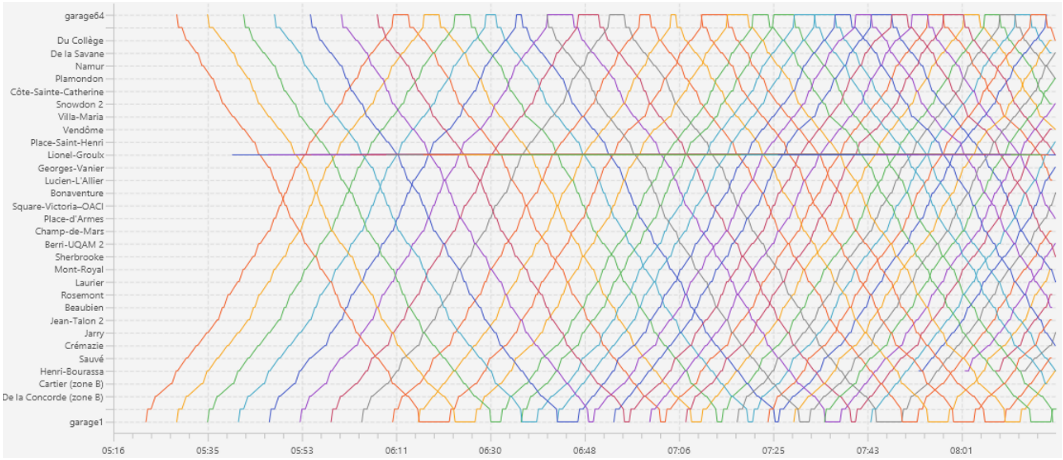


Figure 5 A Space-Time diagram for the Orange line.

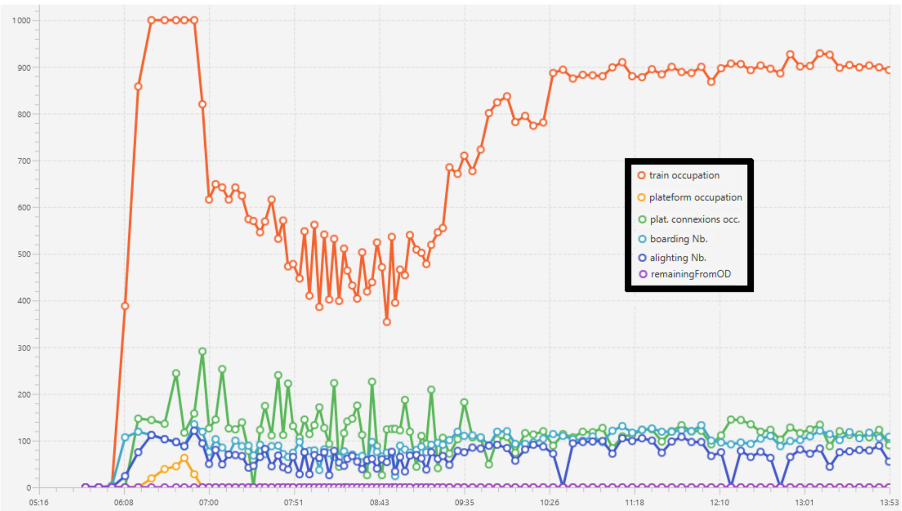


Figure 6 Passenger flows at Berri-Uqam Orange line.

Figure 6 gives information on passenger flows at station Berri-UQAM (BUQ), more precisely for the platform located on the orange line. This station is an important node in the network, because it is an interconnection between the orange and green lines. The horizontal axis represents time, and the vertical axis a number of passengers. Each point on each curve was recorded at the date of closure of train doors, i.e. when a train is about to leave the station. The orange curve describes occupation of trains that stop at the considered platform. As one can see the trains rapidly fill at their maximum capacity between 6:00 and 7:00. The green curve represents passengers who alighted at BUQ, and will move to the green line in the next coming seconds, the light blue curve represents the number of passengers that boarded the currently stopped train. The dark blue curve represents the number of passengers who alighted at that station but do not connect to the green line (they will exit the station). Last, the yellow curve represents passengers who failed to board because the train was full. This is an important indicator, that impacts user satisfaction. The pink curve depicts the number of passengers who had to wait before entering the station (this value remains low during the depicted simulation because platforms are never full).

The simulation is done with heavy passenger flows conditions from early morning until 12 o'clock. The consequence is that trains are almost full on the whole time period shown in Figures 5 and 6 except for the period 07:00 - 10:00, when frequency of trains is increased sufficiently to reduce platforms occupation and hence avoid failures to board. Then, from 7:00 to 9:30AM, train occupation is low to 50-60% and no passenger fails to baord. After 10:00AM, the number of trains on the network is reduced. As the incoming passenger flows remain high on all lines, trains occupation increases. However frequency of stops remains sufficient to keep trains occupation at 90% of their maximal capacity, and the number of boarding and alighting passengers remains rather stable after 10:00 during the simulation.

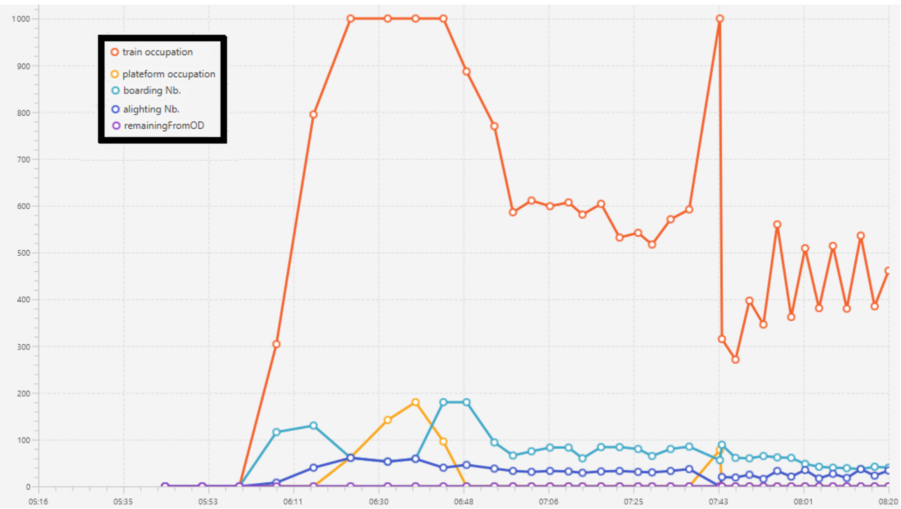


Figure 7 A delay at station Sauve creates a passengers peak at Beaubien.

In a second experiment, we studied the consequence of a 10min delay at 7:30AM at station Sauvé (the 5^{th} station on the Orange line) with the same weekday scenario. For space reasons, the space-time diagram of this scenario appears in Appendix I. The networks takes around 30 minutes to recover from this primary delay. Figure 7 shows the consequences of this perturbation on passengers at station Beaubien (located 4 stations after Sauvé on the orange line). A peak in trains occupation arises at 7:43AM. It is due to the increased time gap between train departures, but also to a larger number of passengers boarding at the previous stations for the same reason.

5 Conclusion

We have proposed a model for metro networks with their passenger flows. This extension of Petri nets includes time, positions of moving objects, and queues to represent passenger moves between platforms and metros. The proposed semantics allows fast simulation, by computing the state of the system at each discrete event occurrence. At each event, the time elapsed is known, which is sufficient to update trajectories and passenger flows. Simulation of a complete day of operation is fast enough to handle real-size cases such as the Metro of Montreal, and study the effects of operational choices on passenger flows.

A first direction to extend this work is to consider other values than the number of passengers. For instance, considering the energy consumed by a metro network is a crucial need. Another challenging task is to address traffic management as a controller synthesis problem, building controllers that optimize a quantitative criterion (energy, nb. of failures to board...). Standard approaches of control (e.g. à la [4]) will not work for metros, due to the number of states to consider. Solutions to build effectively optimized traffic management algorithms may come from abstraction, approximation, and possibly learning techniques. Last, even if this model is tailored for metros, the concurrent nature of the model should be useful in a setting where objects are most of the time independent, and interact only in local delimited areas, such as road sections or automated plants.

References

1 Y. Ahn, T. Kowada, H. Tsukaguchi, and U. Vandebona. Estimation of passenger flow for planning and management of railway stations. *Transportation Research Procedia*, 25:315–330, 2017. World Conference on Transport Research - WCTR 2016 Shanghai. 10-15 July 2016. doi:10.1016/j.trpro.2017.05.408.

2 R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

3 N. Bertrand, B. Bordais, L. Hélouët, T. Mari, J. Parreaux, and O. Sankur. Performance evaluation of metro regulations using probabilistic model-checking. In *Proc. of RSSRail 2019*, volume 11495 of *Lecture Notes in Computer Science*, pages 59–76, 2019.

4 C.G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems, Third Edition*. Springer New York, 2021. doi:10.1007/978-0-387-68612-7.

5 T. H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 2nd edition, 2001.

6 Société des Transports de Montréal. GTFS Static Overview. https://www.stm.info/sites/default/files/media/Stminfo/images/plan-metro-blanc.pdf, 2024.

7 Rail Management Consultants International GmbH. Railsys suite. https://www.rmcon-int.de/home-en/.

8 Google. GTFS Static Overview. https://developers.google.com/transit/gtfs, 2022.

9 S. Guanghui, S. Bingfeng, Z. Kun, Z. Ben, and Z. Xuanchuan. Simulation-based method for the calculation of passenger flow distribution in an urban rail transit network under interruption. *Urban Rail Transit*, 9:110–126, 2023. doi:10.1007/s40864-023-00188-z.

10 R. Haehn. *Optimisation and Analysis of Railway Timetables under Consideration of Uncertainties*. PhD thesis, Aachen University, 2022.

11 R. Haehn, E. Ábrahám, and N. Kotowski. Acceleration techniques for symbolic simulation of railway timetables. In *Proc. of RSSRail'22*, volume 13294 of *Lecture Notes in Computer Science*, pages 46–62, 2022.

12 N. Harris. Train boarding and alighting rates at high passenger loads. *Journal of Advanced Transportation*, 40:249–263, June 2006. doi:10.1002/atr.5670400302.

13 L. Hélouët and P. Agrawal. Waiting nets. In *Proc. of PETRI NETS'22*, volume 13288 of *Lecture Notes in Computer Science*, pages 67–89. Springer, 2022.

14 L. Hélouët and A. Thébault. Mochy: A tool for the modeling of concurrent hybrid systems. In *Application and Theory of Petri Nets and Concurrency*, pages 205–216, 2023.

15 K. Kecir. *Performance evaluation of urban rail traffic management techniques*. PhD thesis, Universite de Rennes 1, 2019.

16 N. Luangboriboon, S. Seriani, and T. Fujiyama. The influence of the density inside a train carriage on passenger boarding rate. *International Journal of Rail Transportation*, 9(5):445–460, 2021.

17 E.W. Mayr. An algorithm for the general petri net reachability problem. *SIAM Journal on Computing*, 13(3):441–460, 1984.

18 P. M. Merlin. *A Study of the Recoverability of Computing Systems*. PhD thesis, University of California, Irvine, CA, USA, 1974.

19 G. Michal, N. Huynh, N. Shukla, A. Munoz, and J. Barthelemy. Railnet: A simulation model for operational planning of rail freight. *Transportation Research Procedia*, 25:461–473, 2017. World Conference on Transport Research - WCTR 2016 Shanghai. 10-15 July 2016. doi:10.1016/j.trpro.2017.05.426.

20 A. Nash and D. Huerlimann. Railroad simulation using OpenTrack. In *Computers in Railways IX*, pages 45–54, 2004.

21 A. Seyfried, T. Rupprecht, O. Passon, B. Steffen, W. Klingsch, and M. Boltes. New insights into pedestrian flow through bottlenecks. *Transportation Science*, 43:395–406, March 2007. doi:10.1287/trsc.1090.0263.

22 STM. STM Developers section. https://www.stm.info/en/about/developers, 2023.

A Computing $\uparrow(tr_{k+1}, TS, h)$

Let $tr_k = \langle(x_0, y_0)(x_1, y_1)\rangle \cdots \langle(x_{n-1}, y_{n-1})(x_n, y_n)\rangle$, and let $d_h \in [x_m, x_{m+1}]$ be the date where $\widehat{tr_{k+1}}(d_h) = tr_k(d_h) + h$. Then

$$\begin{aligned}\uparrow(tr_{k+1}, TS, h) = \;& \langle(0, H).(d_h, tr_k(d_h) + h)\rangle \cdot \langle(d_h, tr_k(d_h) + h)(x_{m+1}, y_{m+1} + h)\rangle \cdots \\ & \cdots \langle(x_{n-1}, y_{n-1} + h)(x_n, y_n + h)\rangle \cdot \langle(x_n, y_n + h)(x_n + h \cdot \tfrac{d}{H}, 0)\rangle\end{aligned}$$

B Formal definition of Left Shift

▶ **Definition 6.** *The* left shift *of a segment* $s = \langle(x_{i-1}, y_{i-1})(x_i, y_i)\rangle$ *is defined if* $x_i \geq \delta$, *and is a segment* $LS(s, \delta) = \langle(x'_{i-1}, y'_{i-1})(x'_i, y'_i)\rangle$, *where :* $x'_{i-1} = \max(0, x_{i-1} - \delta)$, $x'_i = x_i - \delta$, $y'_{i-1} = y_{i-1} + \delta \cdot \frac{y_i - y_{i-1}}{x_i - x_{i-1}}$, $y'_i = y_i$.

Let (tr, b) *be a trajectory with* $tr = s_1 \cdot s_2 \cdots s_k$, *and let* $\delta \leq x_k$. *The left shift of* (tr, b) *by duration* δ, *is denoted* $LS(tr, b)$. *If* b *is true, then* $LS(tr, b) = (tr, b)$. *If* b *is false, then* $LS(tr, b) = LS(S_{i_\delta}, \delta) \cdots LS(s_k, \delta)$, *where* i_δ *the index of the first segment such that* $LS(s_i, \delta)$ *is defined.*

Notice that as soon as $\delta \leq x_k$, index i_δ exists.

C Computing $\delta_{arr}, \delta_{block}$

Let us now detail, for a given configuration $C = (M, \mu)$ and a given place p how to compute the value of $\delta_{arr}(p)$ and $\delta_b lock(p)$ when $\mu(p) = tr_1 \cdot tr_2 \cdots tr_k$. First, the arrival date to consider is the date of the first unblocked trajectory. Let i be the index of this trajectory, and let $tr_i = s_1 \dots s_q$ with $s_q = \langle(x_{q-1}, y_{q-1})(x_q, 0)\rangle$. Then, the train represented by trajectory tr_i can only arrive in station within x_q time units, so $\delta_{arr}(p) = x_q$. In the same setting, one can compute the remaining time before train represented by tr_i has to brake to maintain a safety headway h. Letting $tr_i^{-1}(y)$ denote the date x at which $traj_i(x) = y$, then the train has to adapt its speed at date $\delta_{block}(p) = tr_i^{-1}\big((n-1) \cdot h\big)$.

D Algorithm to compute passenger paths

Algorithm 1 Floyd-Warshall Distance Matrix.

```
parameter $N$ : the set of ids of the places
the cells of the matrix are initially null
$M_D$ : the distance square matrix from an origin to a destination
for all $(id_1, id_2) \in A$ do                              ▷ Matrix initialization
    $M_D(id_1, id_2) = 1$                                    ▷ arcs weights are set to 1
end for
for all $k \in N$ do                                         ▷ Distance Matrix Building
    for all $i \in N$ do
        for all $j \in N$ do
            $v \leftarrow null$
            if $M_D(i,j)! = null$ then
                if $M_D(i,k)! = null$ AND $M_D(k,j)! = null$ then
                    $v \leftarrow min(M_D(i,j), M_D(i,k) + M_D(k,j))$
                else
                    $v \leftarrow M_D(i,j)$
                end if
            else if $M_D(i,k)! = null$ AND $M_D(k,j)! = null$ then
                $v \leftarrow M_D(i,k) + M_D(k,j)$
            end if
            $M_D(i,j) \leftarrow v$
        end for
    end for
end for
```

E Algorithm to compute the next station on a trip

Algorithm 2 Floyd-Warshall Next step Matrix.

```
parameter $M_D$ : from Floyd-Warshall Distance Matrix algorithm
parameter $N$ : the set of ids of the places
the cells of the matrix are initially null
$M_{NS}$ : the next step squared matrix from an origin to a destination
for all $(id_1, id_2) \in A$ do                          ▷ Matrix initialization
    $M_{NS}(id_1, id_2) = id_2$
end for
for all $k \in N$ do                                     ▷ Next Step Matrix Building
    for all $i \in N$ do
        for all $j \in N$ do
            $v \leftarrow null$
            if $i! = j$ then
                if $M_D(i,k) == null$ OR $M_D(j,k) == null$ then
                    $v \leftarrow M_{NS}(i,j)$
                else if $M_D(i,j) == null$ then
                    $v \leftarrow M_{NS}(i,k)$
                else if $M_D(i,j) \leq M_D(i,k) + M_D(k,j)$ then
                    $v \leftarrow M_{NS}(i,j)$
                else
                    $v \leftarrow M_{NS}(i,k)$
                end if
            end if
            $M_{NS}(i,j) \leftarrow v$
        end for
    end for
end for
```

F Complex network patterns with Trajectory nets

Consider for instance Figure 8-a). This piece of trajectory net represent a typical pattern to design forks. In this Figure, a train represented by a segment in place $P_{T,A}$ can be guided to a track U (represented by place $p_{T,U}$) if place $p_{c,U}$ is filled or to track D (represented by place $p_{T,D}$) if place $p_{c,D}$ is filled. Figure 8-b) represents another typical pattern appearing in networks, namely the en of track and the turn back procedure. Form these two examples, one can see that filling control places at the right moment is a way to control arrivals, departures and directions of trains. If timetable is provided, one can even implement it with a controller that fills the appropriate places at the planned departure dates. A pattern of the form of the net given in Figure 9 can also be used to represent a garage, initially filled with a sequence of trajectories representing the available fleet of vehicles.

G Uptading queue, platform and train payloads

Let us detail how passengers waiting on quays, walking, or entered in a train are updated between date d and $d + \delta$. Let $\big(C, d, (P_i)_{i \in 1..K_P}, (TP_i)_{i \in 1..K_T}, (M_{Q_i})_{i \in 1..K_Q}\big)$ be the current configuration, and $Scen = OD_1 \cdots OD_K$ be the scenario used for simulation.

Figure 8 Fork from section A to sections U or D controlled by place $p_{c,U}$ and $p_{c,D}$.

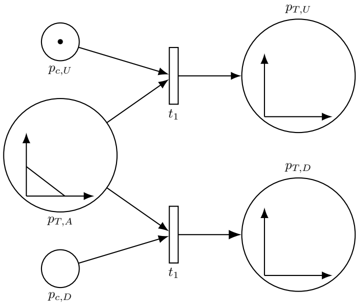
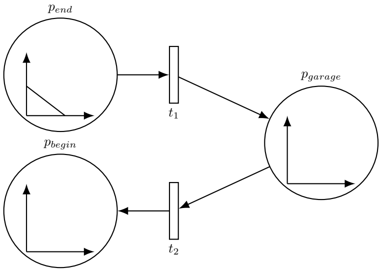


Figure 9 Garage and Reversals.



Let P_i be the contents of a platform at a station S_i, and $Q_i = (l_i, v_i)$ be the queue relating gates at the platform at station S_i. Let M_{Q_i} denote the state of Q_i.

The new state of M_{Q_i} in C' after δ time units is $M'_{Q_i} = M_{Q_i}^{\leftarrow\delta} + Arr_{M_{Q_i}}(Scen, d, d+\delta)$, where $M_{Q_i}^{\leftarrow\delta}$ is the left shift of queue contents, i.e. $M_{Q_i}[\lfloor y-\delta\rfloor \cdot v_i] = M_{Q_i}[y-\delta\cdot v_i]$, and $Arr_{M_{Q_i}}(Scen, d, d+\delta)$ is the payload representing passengers arrived on platform S_i between date d and $d+\delta$.

Formally, let k_1, k_2 be two integers such that $d \in [(k_1-1)\cdot\Delta, k_1\cdot\Delta]$ and $d+\delta \in [(k_2-1)\cdot\Delta, k_2\cdot\Delta]$. Then

$$Arr_{M_{Q_i}}(Scen, d, d+\delta)[x] = \begin{cases} OD[k_2]\cdot\frac{1}{\Delta} \text{ if } x \in [0, d+\delta-k_2] \\ OD[k_2-i]\cdot\frac{1}{\Delta} \text{ if } x \in [d+\delta-k_2+1, d+\delta-k_2-\Delta\cdot i] \\ OD[k_1]\cdot\frac{1}{\Delta} \text{ if } x \in [(k2-k1-1)\cdot\Delta, \delta] \end{cases}$$

Let us now consider the contents of platform P_i when no train is at station S_i. The payload of the platform is incremented by the number of passengers arrived from the entry gate plus queues arriving on the platform. That is, $P'_i[k] = \sum_{x\in 0..\delta} M_{Q_i}[x][k] \sum_{SQ_{j\rightarrow i}} \sum_{x\in 0..\delta} M_{Q_{j,i}}[x][k]$ where $SQ_{j\rightarrow i}$ is the set of queues from a platform j to a platform j.

Let us now consider the situation where a train with payload TP_m is stopped at station S_i We consider a simplified model for exchanges, where passengers board a train when all passengers leaving at station S_i have left the train. We consider that leaving a train takes a constant time Δ_{lt} per passenger, and similarly that boarding takes time Δ_{bt}.

If $TP_m[S_i].\Delta_{lt} > \delta$ then $TP'_m[S_j] = TP_m[S_j]$ (i.e., if the time needed to unload the train at station S_i is larger than δ) then for every $S_j \neq S_i$ and $TP'_m[S_i] = TP_m[S_i] - \lfloor\delta/\Delta_{lt}\rfloor$.

If $TC(p)[S_i].\Delta_{lt} \leq \delta$ then $TP'_m[S_i] = 0$, then the allowed number of passengers after all passengers stopping at S_i have exited the train is $maxboard = TC_{max} - \sum_{S_j\neq S_i} TP_m[S_j]$. The number of passengers who will board also depends on the remaining available time, i.e., $\delta_b = \delta - TP_m[S_i]\cdot\Delta_{lt}$.

Then, the number of passengers boarding who will board is: $nbp_{C,\delta} = \max(\frac{\delta_b}{\Delta_{bt}}, maxboard)$

We divide the stations in two groups: $S^{=}$ containing stations S_k such that $P_i[S_k] < nbp_{C,\delta}/|\mathcal{S}|-1$, and S^{+}, containing stations such that $P_i[S_k] \geq nbp_{C,\delta}/|\mathcal{S}|-1$ For every station $S_k \in S^{=}$ we set $P'_i[S_k] = 0$ and $TP'_m[S_k] = TP_m[S_k] + P_i[S_k]$. The remaining number of passengers is $nbpr_{C,\delta} = nbp_{C,\delta} - \sum_{S_k \in S^{=}} P_i[S_k]$ This number of passengers is fairly distributed on other destination, [2], by allowing n_k passengers to board, with $n_k = \lfloor nbpr_{C,\delta}/|S^{+}| \rfloor$ or $n_k = \lfloor nbpr_{C,\delta}/|S^{+}| \rfloor + 1$, i.e. setting $P'_i[S_k] = P_i[S_k] - n_k$ and $TP'_m[S_k] = TP_m[S_k] + n_k$.

H Map of the metro network in Montreal

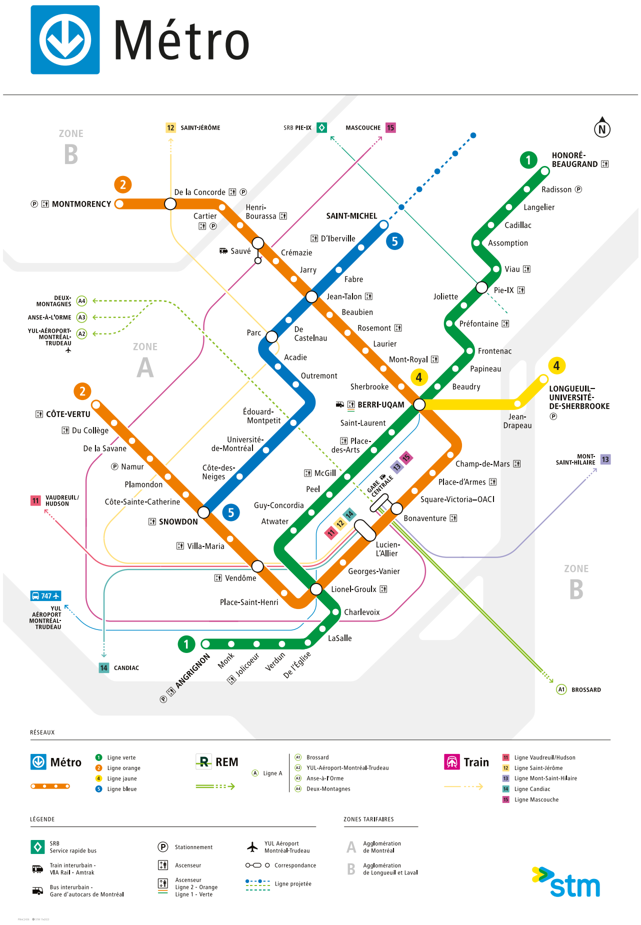


Figure 10 The STM Montreal Network.

[2] To distribute fairly these passengers, we memorize the last destinations with the most passengers, or choose randomly at each round which destinations receive an additional passenger.

I Appendix: A space-time diagram for a delay occurring at station Sauve

The Space-Time diagram below in Figure 11 depicts a incident causing a 10 minutes delay at 7:30AM at station Sauvé during a weekday scenario. Station Sauvé is the 5^{th} station on the Orange line, after a start in station Montmorency. It is located 4 stations before station Beaubien. Figure 11 shows that this incident impacts 4 vehicles. The networks takes around 30 minutes to recover from this primary delay. This can be observed as a "hole" in the parallel lines representing train trips.

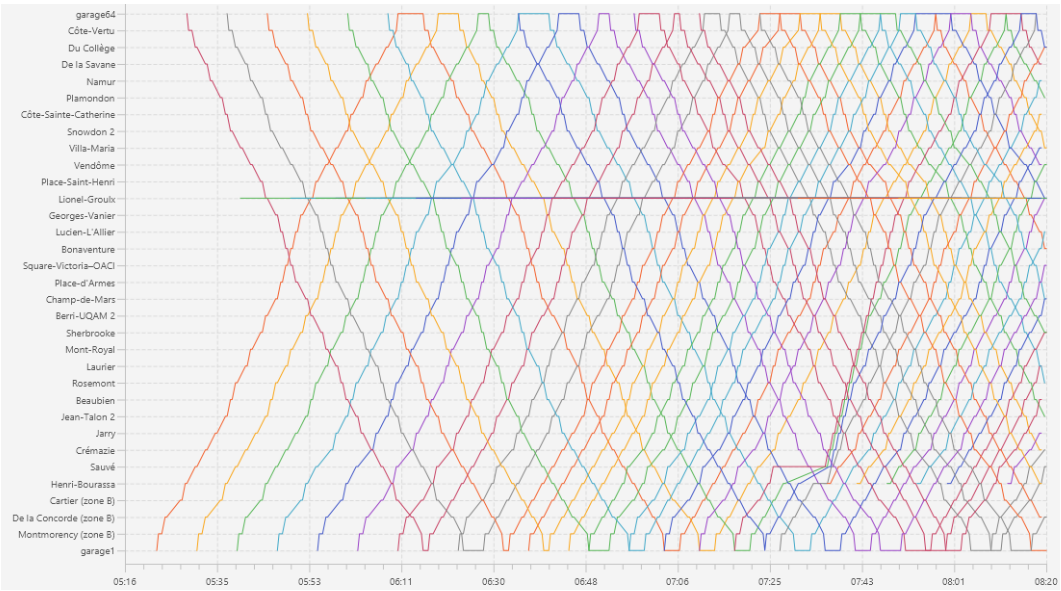


Figure 11 A delay at station Sauve impacts multiple vehicles.

Computing User Equilibria for Schedule-Based Transit Networks with Hard Vehicle Capacities

Tobias Harks
University of Passau, Germany

Sven Jäger
RPTU Kaiserslautern-Landau, Germany

Michael Markl
University of Passau, Germany

Philine Schiewe
Aalto University, Finland

Abstract

Modelling passenger assignments in public transport networks is a fundamental task for city planners, especially when deliberating network infrastructure decisions. A key aspect of a realistic model for passenger assignments is to integrate selfish routing behaviour of passengers on the one hand, and the limited vehicle capacities on the other hand. We formulate a side-constrained user equilibrium model in a schedule-based time-expanded transit network, where passengers are modelled via a continuum of non-atomic agents that want to travel with a fixed start time from a user-specific origin to a destination. An agent's route may comprise several rides along given lines, each using vehicles with hard loading capacities. We give a characterization of (side-constrained) user equilibria via a quasi-variational inequality and prove their existence by generalizing a well-known existence result of Bernstein and Smith (Transp. Sci., 1994). We further derive a polynomial time algorithm for single-commodity instances and an exact finite time algorithm for the multi-commodity case. Based on our quasi-variational characterization, we finally devise a fast heuristic computing user equilibria, which is tested on real-world instances based on data gained from the Hamburg S-Bahn system and the Swiss long-distance train network. It turns out that w.r.t. the total travel time, the computed user-equilibria are quite efficient compared to a system optimum, which neglects equilibrium constraints and only minimizes total travel time.

2012 ACM Subject Classification Theory of computation → Network games; Applied computing → Transportation; Theory of computation → Exact and approximate computation of equilibria

Keywords and phrases traffic assignment, side-constrained equilibrium, public transportation

Digital Object Identifier 10.4230/OASIcs.ATMOS.2024.17

Related Version *Full Version*: `https://arxiv.org/abs/2406.17153` [19]

Supplementary Material
Software (Source Code): `https://github.com/ArbeitsgruppeTobiasHarks/sbta/tree/atmos` [20]
archived at `swh:1:dir:a3e48de8912d81166b2517c3c826e11bcda2c356`

Funding This research was funded by the German Federal Ministry of Education and Research (BMBF). FK 05M22UKB, 05M22WPA.

Acknowledgements We thank the anonymous reviewers for their constructive feedback.

1 Introduction

In the domain of public transport, models describing the assignment of passengers over a transit network are crucial for infrastructure planners to understand congestion phenomena and possible investments into the infrastructure. The existing approaches can roughly be

24th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2024).
Editors: Paul C. Bouman and Spyros C. Kontogiannis; Article No. 17; pp. 17:1–17:17
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

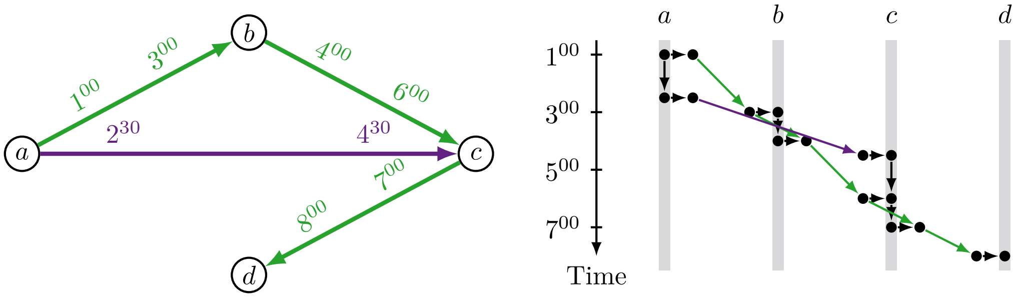


Figure 1 Two scheduled vehicle trips in the physical network (left) and their representation in the time-expanded transit network (right).

categorized into *frequency-based* and *schedule-based* models, see [13, 12] for a survey. The former model class operates operate with line frequencies and implicitly defines resulting travel times and capacities of lines and vehicles, cf. [38, 11, 40, 3, 6, 5, 24]. With variations in the demand profile during peak hours, the frequency-based approach only leads to approximate vehicle loads, with the error increasing as variability grows. In contrast, schedule-based approaches are more fine-grained and capable of explicitly modelling irregular timetables of lines. They are usually based on a *time-expanded transit network* derived from the physical transit network and augmented by (artificial) edges such as waiting, boarding, alighting, dwelling, and driving edges to connect different stations. This construct, also known as diachronic graph [29] or space-time network [4], is illustrated in Figure 1.

An assignment of passengers to paths in this network encompasses their entire travel strategies, including line changes, waiting times, etc. It corresponds to a path-based multi-commodity network flow satisfying all demand and supply. One key obstacle in the analysis of such a schedule-based model is the integration of *strategic behaviour* of passengers, opting for shortest routes, and the *limited vehicle capacity*, which bounds the number of passengers able to use a vehicle at any point in time. If a vehicle is already at capacity, further passengers might not be able to enter this vehicle at the next boarding stop, which can make their (shortest) route infeasible. On the other hand, the passengers already in the vehicle are not affected by the passengers wishing to board.

A key issue of such a capacitated model is to choose the right equilibrium concept. Consider for instance the simple example in Figure 1, and suppose that the vehicles operating the violet and the green line have a capacity of 1 unit each. A demand volume of 2 units start their trip at node a at time 1^{00}, and all particles want to travel as fast as possible to the destination node c. The violet line arrives at 4^{30}, while the green line arrives at 6^{00}. Then there exists no capacity-feasible *Wardrop equilibrium* [39, 8], i.e., a flow only using quickest paths.

Most works in the literature deal with this non-existence by either assuming soft vehicle capacities (cf. [9, 30, 28]) or by considering more general travel strategies and a probabilistic loading mechanism (cf. [26, 23, 27, 17]). An alternative approach that inherently supports capacities are so-called *side-constrained user equilibria*. These are feasible flows such that for any used path there is no *available alternative path* with lower cost (cp. [10]). The precise definition of what is an available lower-cost alternative has spurred a whole series of works. Larsson and Patriksson [32, 25] consider only paths with available residual capacity as available alternative paths. This excludes the change from a used path p to another path q that shares some saturated edges with p. For a path p, Smith [37] considers all paths q to which a part of the flow on p can change over so that the resulting flow is feasible.

This concept has the drawback that it allows for coordinated deviations of bundles of users, which is unrealistic and, as shown by Smith, leads to non-existence of such side-constrained equilibria for monotonic, continuous, but non-separable cost functions. In response, Bernstein and Smith [2] propose an alternative equilibrium concept (see Definition 3 of *BS-equilibrium*) characterized by the property that "no arbitrarily small bundle of drivers on a common path can strictly decrease its cost by switching to another path" [7].

In this paper, we consider side-constrained user equilibria for schedule-based time-expanded networks in the sense of Bernstein and Smith [2]. Here, whether a path is an available alternative depends only on the available capacity of the vehicle when the passenger boards it, but not on whether capacity is exceeded on a later edge of the vehicle trip. Hence, a path can be an available alternative for some user even if arbitrarily small deviations to that path make the resulting flow infeasible (for some other users). Similar to [28], the priority of passengers in the vehicle can be modelled by expressing the capacity limitations using discontinuous costs on the boarding edges in the time-expanded transit network. The resulting cost map is not separable, and it turns out that it does not satisfy the regularity conditions imposed by Bernstein and Smith [2] to prove existence of equilibria. Our approach works for elastic demands where a user only travels if the travel cost does not exceed the user's willingness to travel. This elastic demand model is quite standard in the transportation science literature, see [41] and references therein.

1.1 Our Contribution

We define a user equilibrium for schedule-based time-expanded networks using the notion of deviations. For a given flow, an *admissible ε-deviation* corresponds to shifting an ε-amount of flow from a path p to another path q without exceeding the capacity of any boarding edge along q. A feasible flow is a *side-constrained user equilibrium* if there are no improving admissible ε-deviations for arbitrarily small ε. We summarize our contribution as follows.

1. We prove that for schedule-based time-expanded networks, side-constrained user equilibria can be characterized by a quasi-variational inequality defined over the set of admissible deviations (Theorem 2). Moreover, by moving the side-constraints into discontinuous cost functions, we can express side-constrained user equilibria as BS-equilibria (Theorem 4).
2. We study the central question of the existence of BS-equilibria. We first give an example showing that our cost map does not fall into the category of *regular* cost maps, defined by Bernstein and Smith [2], for which they showed the existence of equilibria. Instead, we introduce a more general condition for cost maps, which we term *weakly regular*. As our main theoretical result, we prove in Theorem 7 that BS-equilibria do exist for weakly regular cost maps. We further show (Theorem 8) that the cost maps in schedule-based time-expanded networks are weakly regular, hence Theorem 7 applies. However, the generalization of Bernstein and Smith's result might also be of interest for other traffic models.
3. We then turn to the computation of BS-equilibria. For single-commodity time-expanded networks, we present an algorithm that computes a BS-equilibrium in quadratic time relative to the number of edges of the input graph (Theorem 12). For multi-commodity networks, we give an exact finite-time algorithm. As this algorithm is too slow for practical computations, we further develop a heuristic based on our quasi-variational inequality formulation. It starts with an arbitrary feasible flow and updates this flow along elementary admissible deviations in the sense of Theorem 2.
4. Finally, we test our heuristic on realistic instances drawn from the Hamburg S-Bahn network and the Swiss long-distance train network. It turns out that user-equilibria can be computed with our heuristic and that they are quite efficient compared to a system

optimum, which neglects equilibrium constraints and minimizes total travel time. More specifically, for the computed instances the total travel times in the user equilibria are less than 8% higher compared to the system optimum.

1.2 Related Work

Schedule-based transit assignment has been studied extensively in the past. Since the works by [29, 4], most authors use a time-expanded graph as their modelling basis. For example, Carraresi et al. [4] consider a model with hard capacity constraints where passengers accept routes that are at most a factor of $1+\varepsilon$ worse than an optimal path without any congestion. This is approximated in several papers [9, 28, 30] by incorporating the vehicle capacities as continuous penalties representing the discomfort experienced by using an overcrowded edge.

Marcotte and Nguyen [26] address hard capacities by defining an agent's *strategy* as preference orderings of outgoing edges at each node, similar to so-called hyperpaths, and by assuming a random loading mechanism for congested edges, where the probability of entering an edge is proportional to its capacity and decreases with the amount of flow desiring to traverse it. Every passenger wants to minimize the *expected* travel cost resulting from their strategy. A whole series of works [28, 17, 16, 15, 33, 18] build upon this model and expand it by incorporating boarding priorities of passengers, departure time choice with early/late arrival penalties, different costs for seated and standing passengers, risk aversion, or stochastic link travel times modelling variation due to weather or incidents.

An alternative to time-expanded graphs is the use of dynamic flows. For example, [31] defines dynamic flows that traverse the public-transport edges in discrete chunks and employ the method of successive averages (MSA) to find approximate equilibria. Side-constrained equilibria for dynamic flows have been studied in [14] where also a dynamic variant of BS-equilibria is analysed without stating any existence results for them.

2 Side-Constrained Equilibria for Schedule-Based Transit Networks

We first describe a schedule-based time-expanded network (cf. [29, 4]) and then formally define the side-constrained user equilibrium concept.

2.1 The Schedule-Based Time-Expanded Network

Consider a set of geographical stations S (e.g., metro stations or bus stops) and a set of vehicle trips Z (e.g., trips of metro trains or buses), specified by their sequence of served stations and adhering to a fixed, reliable timetable. This timetable specifies the arrival and departure times at all stations of the trip, where the arrival time at a station is always strictly later than the departure time of the previous station. Each vehicle trip $z \in Z$ also has an associated capacity ν_z which represents the maximum number of users the corresponding vehicle may hold at any time. We use the term *vehicle* synonymously with *vehicle trip*.

To represent the passengers' routes through the network, we construct a time-expanded, directed, acyclic graph $G = (V, E)$ with a time $\theta(v) \in \mathbb{R}$ assigned to each node $v \in V$.

There are three categories of nodes: an *on-platform node* for each station $s \in S$ and each time θ at which at least one vehicle departs or arrives in s, a *departure node* for each vehicle $z \in Z$ and each time θ at which z departs from a station s, and an *arrival node* for each vehicle $z \in Z$ and each time θ at which z arrives at a station s.

There are five categories of edges connecting these nodes: A *waiting edge* connects two on-platform nodes v, w of the same station s with consecutive times $\theta(v) < \theta(w)$. A *boarding edge* connects an on-platform node with a departure node of a vehicle z of common time θ

and station s. A *driving edge* connects a departure node with the next stop's arrival node of the shared vehicle z. An *alighting edge* connects an arrival node of a vehicle z with the on-platform node of common time θ and station s. Finally, a *dwelling edge* connects an arrival node of a vehicle z with the corresponding departure node at the same station s.

For ease of notation, let E_B and E_D denote the set of all boarding and driving edges, respectively. We denote the time delay of an edge $e = vw$ by $\tau_e := \theta(w) - \theta(v)$, the delay of a v-w-path $p = (e_1, \ldots, e_k)$ by $\tau_p := \sum_{e \in p} \tau_e = \theta(w) - \theta(v)$. For a driving edge $e \in E_D$ belonging to a vehicle $z \in Z$, we write $\nu_e := \nu_z$. Waiting and driving edges are always time-consuming, dwelling edges may be time-consuming, and boarding and alighting edges are instantaneous. For $e \in E_B$, we denote the succeeding driving edge by e^+.

Figure 1 shows a possible generated graph for two vehicles, a green one and a violet one, and four stations a, b, c, and d. The nodes on the grey rectangles represent the on-platform nodes, the other nodes are the departure and arrival nodes.

The non-atomic users of the network are partitioned into several groups: First, they are grouped into finitely many origin-destination and departure-time triplets $(s_j, t_j, \theta_j) \in \mathcal{T} \subseteq S \times S \times \mathbb{R}$. To model the elastic demand, there is a non-increasing function $Q_j : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ for each triplet that given some travel time τ assigns the volume of particles of triplet j that are willing to travel at cost τ. We assume $Q_j(\tau) = 0$ for all $\tau \geq T$ for some $T \in \mathbb{R}$. Let $\mathcal{P}_j^\circ$ denote the set of paths from the on-platform node at time θ_j and station s_j to any on-platform node of station t_j.

As the travel times of the paths are fixed, we can subdivide all triplets into a finite number of *commodities* I of common willingness to travel and introduce an outside option for each commodity: Let $\{\tau_{j,1}, \ldots, \tau_{j,k_j}\} = \{\tau_p \mid p \in \mathcal{P}_j^\circ\}$ be the set of travel times of all paths $p \in \mathcal{P}_j^\circ$ ordered by $\tau_{j,1} < \cdots < \tau_{j,k_j}$. For each $j' \in \{1, \ldots, k_j + 1\}$, we introduce a commodity $i_{j,j'}$ consisting of all particles of triplet j whose willingness to travel is contained in the interval $[\tau_{j,j'-1}, \tau_{j,j'})$ with $\tau_{j,0} := 0$ and $\tau_{j,k_j+1} := T$. Thus, commodity $i := i_{j,j'}$ has a demand volume of

$$Q_i := Q_j(\tau_{j,j'-1}) - Q_j(\tau_{j,j'}),$$

and we assign it an outside option p_i^{out} with some constant cost $\tau_{p_i^{\text{out}}}$ chosen from $(\tau_{j,j'-1}, \tau_{j,j'})$. Finally, we write $\mathcal{P}_i := \mathcal{P}_j^\circ \cup \{p_i^{\text{out}}\}$, and denote the set of all commodity-path pairs by $\mathcal{P} := \{(i,p) \mid p \in \mathcal{P}_i\}$.

2.2 Side-Constrained User Equilibrium

A *(path-based) flow* f is a vector $(f_{i,p})_{(i,p) \in \mathcal{P}}$ with $f_{i,p} \in \mathbb{R}_{\geq 0}$. We call the flow f

- *demand-feasible*, if $\sum_{p \in \mathcal{P}_i} f_{i,p} = Q_i$ holds for all $i \in I$,
- *capacity-feasible*, if $f_e := \sum_{i,p : e \in p} f_{i,p} \leq \nu_e$ holds for all driving edges $e \in E_D$,
- *feasible*, if f is both demand- and capacity-feasible.

Let $\mathcal{F}_Q$, $\mathcal{F}^\nu$, $\mathcal{F}_Q^\nu$ denote the sets of all demand-feasible, capacity-feasible, and feasible flows, respectively. For a given demand-feasible flow f and two paths $p, q \in \mathcal{P}_i$ with $f_{i,p} \geq \varepsilon$, we define the ε*-deviation* from p to q

$$f_{i,p\to q}(\varepsilon) := f + \varepsilon \cdot (1_{i,q} - 1_{i,p})$$

as the resulting flow when shifting an ε-amount of flow of commodity i from p to q. We call $f_{i,p\to q}(\varepsilon)$ an *admissible* deviation, if $f_{i,p\to q}(\varepsilon)_{e^+} \leq \nu_{e^+}$ holds for all boarding edges e of q. We say, q is an *available alternative* to p for i with respect to f, if there is some positive ε such that $f_{i,p\to q}(\varepsilon)$ is an admissible deviation. In other words, q is an available alternative, if after switching some arbitrarily small amount of flow from p to q, the path q does not involve boarding overcrowded vehicles. Equivalently, all boarding edges $e \in q$ fulfil $f_{e^+} < \nu_{e^+}$, if $e^+ \notin p$, and $f_{e^+} \leq \nu_{e^+}$, if $e^+ \in p$. We denote the set of available alternatives to p for i with respect to f by $A_{i,p}(f)$.

▶ **Definition 1.** *A feasible flow f is a* (side-constrained) user equilibrium *if for all $i \in I$ and $p \in \mathcal{P}_i$ the following implication holds:*

$$f_{i,p} > 0 \implies \forall q \in A_{i,p}(f) : \tau_p \leq \tau_q.$$

This means, a feasible flow is a side-constrained user equilibrium if and only if a path is only used if all its faster alternative routes are unavailable due to the boarding capacity constraints. For the rest of this work, we use the shorthand *user equilibrium* for this concept.

3 Characterization and Existence

We characterize user equilibria as defined above in two different ways: as solutions to a quasi-variational inequality and as BS-equilibria by defining appropriate, discontinuous cost functions. By generalizing the existence result of Bernstein and Smith [2] for BS-equilibria, we show that user equilibria exist. Some proofs are omitted in this conference paper due to space constraints. They can be found in the full version [19] of the paper.

3.1 Quasi-Variational Inequalities

Traditional types of user equilibria without hard capacity constraints can be equivalently formulated as a solution to a variational inequality of the form

$$\text{Find } f^* \in D \text{ such that:} \qquad \langle c(f^*), f - f^* \rangle \geq 0 \quad \text{for all } f \in D, \qquad (\mathrm{VI}(c, D))$$

where D is a closed, convex set, and c is a continuous cost function.

With the introduction of hard capacity constraints together with boarding priorities, an admissible ε-deviation might lead to capacity violations. Therefore, such deviations may leave the feasible set $D = \mathcal{F}_Q^\nu$ and, thus, are not representable in this variational inequality, leading us to the concept of quasi-variational inequalities. We define the set-valued function

$$M : \mathcal{F}_Q^\nu \rightrightarrows \mathbb{R}_{\geq 0}^{\mathcal{P}}, \quad f \mapsto \{f_{i,p\to q}(\varepsilon) \mid f_{i,p\to q}(\varepsilon) \text{ is an admissible } \varepsilon\text{-deviation}, \varepsilon > 0\}$$

that returns for any given flow f the set of all possible flows obtained by any admissible ε-deviation with respect to f. We now consider the following quasi-variational inequality:

$$\text{Find } f^* \in \mathcal{F}_Q^\nu \text{ such that:} \qquad \langle (\tau_p)_{i,p}, f - f^* \rangle \geq 0 \quad \text{for all } f \in M(f^*). \qquad (\mathrm{QVI})$$

A feasible flow f is a solution to this quasi-variational inequality if and only if there is no commodity i and a pair of paths $p, q \in \mathcal{P}_i$ with $\tau_q < \tau_p$ such that $f_{i,p\to q}(\varepsilon)$ is an admissible ε-deviation for arbitrary $\varepsilon > 0$. Hence, we can characterize user equilibria as follows:

▶ **Theorem 2.** *A feasible flow f^* is a user equilibrium if and only if it is a solution to the quasi-variational inequality* (QVI)*.*

While the existence of solutions to customary variational inequalities in the form of (VI(c, D)) can be shown using Brouwer's fixed point theorem, the existence of solutions to quasi-variational inequalities is not clear upfront. To establish an existence result, we therefore introduce an alternative characterization of our problem in the next section.

3.2 Bernstein-Smith Equilibrium

In this section, we will reformulate the user equilibrium as a Bernstein-Smith equilibrium for suitably chosen edge cost functions $c_e \colon \mathcal{F}_Q \to \mathbb{R}_{\geq 0}$. This way we dispense with the explicit side-constraints and instead incorporate them as discontinuities into the cost functions, so that any equilibrium must correspond to a feasible flow.

▶ **Definition 3** ([2, Definition 2]). *We are given a directed graph $G = (V, E)$ and a set of commodities I, each equipped with a demand $Q_i \geq 0$, a finite, non-empty set of paths $\mathcal{P}_i$, and a cost function $c_{i,p} \colon \mathbb{R}^{\mathcal{P}}_{\geq 0} \to \mathbb{R}_{\geq 0}$ for every $p \in \mathcal{P}_i$. A demand-feasible flow $f \in \mathcal{F}_Q$ is a* Bernstein-Smith equilibrium (BS-equilibrium) *if, for all $i \in I$ and $p \in \mathcal{P}_i$, $f_{i,p} > 0$ implies $c_{i,p}(f) \leq \min_{q \in \mathcal{P}_i} \liminf_{\varepsilon \downarrow 0} c_{i,q}(f_{i,p \to q}(\varepsilon))$.*

We first define cost functions for the edges of our schedule-based transit network as follows: The cost of a non-boarding edge e is given by the time it takes to traverse the edge, i.e., $c_e(f) \coloneqq \tau_e \geq 0$. Passing a boarding edge takes no time, however, it is only possible to board until the capacity of the vehicle is reached. We realize this by raising the cost of the boarding edge when the capacity is exceeded to a sufficiently large constant M, which is guaranteed to be higher than the cost of any available path, e.g., $M \coloneqq \max_{i \in I, p \in \mathcal{P}_i} \tau_p + 1$. This means, for a boarding edge $e \in E_B$, the experienced cost is

$$c_e(f) \coloneqq \begin{cases} 0, & \text{if } f_{e^+} \leq \nu_{e^+}, \\ M, & \text{if } f_{e^+} > \nu_{e^+}. \end{cases} \tag{1}$$

For a v-w-path p in our time-expanded graph, we assign the cost

$$c_{i,p}(f) \coloneqq \sum_{e \in p} c_e(f) = \theta(w) - \theta(v) + \sum_{e \in p \cap E_B} c_e(f). \tag{2}$$

For the outside options p_i^{out}, we assume that they are virtual paths consisting of a single edge in E, which is exclusively used by the path p_i^{out}, and we set the cost of this edge to $c_{i,p_i^{\text{out}}}(f) \coloneqq \tau_{p_i^{\text{out}}}$. The BS-equilibria with respect to these cost functions are exactly the user equilibria, as the following theorem shows.

▶ **Theorem 4.** *A demand-feasible flow f is a user equilibrium if and only if it is a BS-equilibrium with respect to the cost functions $c_{i,p}$ defined above.*

Proof. For any $f \in \mathcal{F}_Q^{\nu}$ and $p, q \in \mathcal{P}_i$ with $f_{i,p} > 0$, it holds that

$$\limsup_{\varepsilon \downarrow 0} c_{i,q}(f_{i,p \to q}(\varepsilon)) \begin{cases} = \tau_q, & \text{if } \forall e \in E_B \cap q : e^+ \in p \text{ or } f_{e^+} < \nu_{e^+}, \\ \geq M, & \text{otherwise.} \end{cases}$$

Note that the condition in the case distinction is equivalent to $q \in A_{i,p}(f)$.

If f is a BS-equilibrium, then $c_{i,p}(f) \leq \tau_{p_i^{\text{out}}} < M$ holds by the BS-equilibrium condition and, thus, we must have $c_{i,p}(f) = \tau_p$. The same holds true, if we instead assume f to be a user equilibrium. Therefore, the following equivalence holds:

$$\forall q \in A_{i,p}(f) : \tau_p \leq \tau_q \iff \forall q \in \mathcal{P}_i : c_{i,p}(f) \leq \limsup_{\varepsilon \downarrow 0} c_{i,q}(f_{i,p \to q}(\varepsilon)).$$

Hence, f is a BS-equilibrium if and only if it is a user equilibrium. ◀

3.3 Existence

Bernstein and Smith [2, Theorem 2] proved the existence of BS-equilibria in the case that each path cost function has the form $c_{i,p} = \sum_{e\in p} c_e$, where $c_e\colon \mathcal{F}_Q \to \mathbb{R}_{\geq 0}$, $e \in E$, are lower-semicontinuous, bounded functions that satisfy the following regularity condition.

▶ **Definition 5** ([2]). *A cost structure $c\colon \mathcal{F}_Q \to \mathbb{R}^E_{\geq 0}$ is* regular *if it satisfies*

$$\liminf_{\varepsilon\downarrow 0} c_{i,q}(f_{i,p\to q}(\varepsilon)) = \sum_{e\in p\cap q} c_e(f) + \sum_{e\in q\setminus p} \bar{c}_e(f)$$

for all $f \in \mathcal{F}_Q$, $i \in I$, and paths $p, q \in \mathcal{P}_i$ with $f_{i,p} > 0$, where $\bar{c}_e$ is the upper hull of c_e defined as

$$\bar{c}_e(f) := \lim_{\varepsilon\downarrow 0} \sup\{c_e(x) \mid x \in \mathcal{F}_Q, \|x - f\| < \varepsilon\}.$$

Unfortunately, the cost functions in our case as defined in (1) do not fulfil this condition, as the network in Figure 1 illustrates: Assume there is a single commodity with source a and sink d with demand 2, and assume that both vehicles have a capacity of one. Let p be the a-d-path using only the green vehicle, and let q be the a-d-path using both vehicles. Let f be the flow sending one unit along p and the remaining unit along the commodity's outside option p_i^{out}. Then, for the boarding edge e of the green vehicle at station c, it holds that $\bar{c}_e(f) = M$ (as $\mathcal{F}_Q$ contains $f_{i,p_i^{\text{out}}\to p}(\varepsilon)$ for $\varepsilon \leq 1$). This implies

$$\liminf_{\varepsilon\downarrow 0} c_q(f_{i,p\to q}(\varepsilon)) = \tau_q < M \leq \sum_{e\in p\cap q} c_e(f) + \sum_{e\in q\setminus p} \bar{c}_e(f).$$

On the left-hand side it is noticed that the flow on the last driving edge is unchanged and boarding remains possible, whereas the right-hand side is oblivious to the flow reduction along p.

Therefore, we introduce a weaker condition that is satisfied in our time-expanded networks but still guarantees the existence of equilibria.

▶ **Definition 6.** *A cost structure $c\colon \mathcal{F}_Q \to \mathbb{R}^E_{\geq 0}$ is called* weakly regular *if the following implication holds for all demand-feasible flows $f \in \mathcal{F}_Q$, $i \in I$, and $p \in \mathcal{P}_i$ with $f_{i,p} > 0$:*

$$c_{i,p}(f) \leq \min_{q\in\mathcal{P}_i} \sum_{e\in p\cap q} c_e(f) + \sum_{e\in q\setminus p} \bar{c}_e(f) \implies c_{i,p}(f) \leq \min_{q\in\mathcal{P}_i} \liminf_{\varepsilon\downarrow 0} c_{i,q}(f_{i,p\to q}(\varepsilon)).$$

It is easy to see that any regular cost structure is also weakly regular.

▶ **Theorem 7.** *If $c\colon \mathcal{F}_Q \to \mathbb{R}^E_{\geq 0}$ is a lower-semicontinuous, bounded, and weakly regular cost structure, a BS-equilibrium exists.*

The proof uses a similar approach to that in [2, Theorem 2], but we show that it applies to the broader class of weakly regular cost structures.

Proof. Let M be an upper bound for all functions c_e. There exists, for each $e \in E$, a sequence of continuous functions $c_e^{(n)}\colon \mathcal{F}_Q \to [0, M]$ such that $c_e^{(n)}(f) \uparrow c_e(f)$ holds for all $f \in \mathcal{F}_Q$. For each $n \in \mathbb{N}$, there is a Wardrop equilibrium $f^{(n)} \in \mathcal{F}_Q$ w.r.t. the path cost function $(c_{i,p}^{(n)})_{(i,p)\in\mathcal{P}}$ defined by $c_{i,p}^{(n)}(f) := \sum_{e\in p} c_e^{(n)}(f)$, i.e., $f_{i,p}^{(n)} > 0$ implies $c_{i,p}^{(n)}(f^{(n)}) \leq c_{i,q}^{(n)}(f^{(n)})$ for all $i \in I$ and paths $p, q \in \mathcal{P}_i$ [36]. Equivalently, we have

$$f_{i,p}^{(n)} > 0 \implies \sum_{e\in p\setminus q} c_e^{(n)}(f^{(n)}) \leq \sum_{e\in q\setminus p} c_e^{(n)}(f^{(n)}). \tag{3}$$

The sequence $(f^{(n)}, c^{(n)}(f^{(n)}))$ is contained in the compact set $\mathcal{F}_Q \times [0, M]^E$, and therefore has a convergent sub-sequence with some limit (f, x); we pass to this sub-sequence.

By the upper-semicontinuity of the upper hull and the monotonicity of the sequence of cost functions, we have for all $e \in E$

$$\bar{c}_e(f) \geq \limsup_{n\to\infty} \bar{c}_e(f^{(n)}) \geq \limsup_{n\to\infty} c_e(f^{(n)}) \geq \lim_{n\to\infty} c_e^{(n)}(f^{(n)}) = x_e. \tag{4}$$

Let $\alpha > 0$. First, since $(c_e^{(n)})_n$ converges pointwise to c_e, there exist $n_0 \in \mathbb{N}$ such that $c_e^{(n_0)}(f) \geq c_e(f) - \alpha/2$. Second, since $c_e^{(n_0)}$ is continuous, there is $\delta > 0$ such that for all $g \in \mathcal{F}_Q$ with $\|f - g\| < \delta$ we have $c_e^{(n_0)}(g) \geq c_e^{(n_0)}(f) - \alpha/2$. As $(c_e^{(n)})_n$ is a pointwise increasing sequence, we then have for all $n \geq n_0$ that $c_e^{(n)}(g) \geq c_e(f) - \alpha$. Third, since $(f^{(n)})_n$ converges to f, there is n_1 such that $\left\|f - f^{(n)}\right\| < \delta$ holds for all $n \geq n_1$. In conclusion, $c_e^{(n)}(f^{(n)}) \geq c_e(f) - \alpha$ holds for $n \geq \max\{n_0, n_1\}$. Since $\alpha > 0$ was arbitrary, we deduce $x_e \geq c_e(f)$.

Let $i \in I$ and $p, q \in \mathcal{P}_i$ with $f_{i,p} > 0$. There exists $n_0 \in \mathbb{N}$ with $f_{i,p}^{(n)} > 0$ for $n \geq n_0$. Taking the limit of (3) yields $\sum_{e\in p\setminus q} x_e \leq \sum_{e\in q\setminus p} x_e$, and applying (4) and $x_e \geq c_e(f)$ we get

$$\sum_{e\in p\setminus q} c_e(f) \leq \sum_{e\in q\setminus p} \bar{c}_e(f).$$

Adding $c_e(f)$ for each $e \in p \cap q$ to both sides , this shows

$$c_p(f) \leq \sum_{e\in p\cap q} c_e(f) + \sum_{e\in q\setminus p} \bar{c}_e(f).$$

Thus, we can apply weak regularity. ◀

While it is clear that the cost functions in (1) are lower-semicontinuous and bounded, some effort is required to show that they fulfil weak regularity. The idea is that given a path p and a path q minimizing $\liminf_{\varepsilon\downarrow 0} c_{i,q}(f_{i,p\to q}(\varepsilon))$ we consider the last common node v of p and q, and define q' as the path that consists of p up until v concatenated with the suffix of q starting from v. For boarding edges e on the second part of q', we can then show $\liminf_{\varepsilon\downarrow 0} c_e(f_{i,p\to q'}(\varepsilon)) = \bar{c}_e(f)$ while boarding edges e on the first part fulfil $\liminf_{\varepsilon\downarrow 0} c_e(f_{i,p\to q'}(\varepsilon)) = c_e(f)$. The observation $\liminf_{\varepsilon\downarrow 0} c_{i,q'}(f_{i,p\to q'}(\varepsilon)) \leq \liminf_{\varepsilon\downarrow 0} c_{i,q}(f_{i,p\to q}(\varepsilon))$ then concludes the argument.

▶ **Theorem 8.** *For schedule-based transit networks, a user equilibrium always exists.*

▶ Remark 9. Even for single-commodity networks, the price of stability, i.e., the ratio of the total travel times in a best user equilibrium and in a system optimum, is unbounded. However, Section 5 shows that this ratio is well-behaved in experiments on real-world networks.

4 Computation of Equilibria

We continue by discussing the computation of user equilibria. After describing an $\mathcal{O}(|E|^2)$ algorithm for single-commodity networks, we consider the multi-commodity case, for which we outline a finite algorithm. To compute multi-commodity equilibria in practice, we propose a heuristic based on insights gained by the characterization with the quasi-variational inequality.

4.1 An Efficient Algorithm for Single-Commodity Networks

We begin with the description of an efficient algorithm for single-commodity networks. To reduce noise, we omit the index i where applicable, i.e., we write $\mathcal{P}$ instead of $\mathcal{P}_i$, etc.

▶ **Definition 10.** *Let $p, q \in \mathcal{P}$. We say that a driving edge $e \in p \cap q$ is a* conflicting edge of p and q *if its corresponding boarding edge e_B lies either on p or on q (but not on both).*

Assume p and q have a conflicting edge, and let $e \in E$ be the first conflicting edge. We say p has priority over q *if the boarding edge e_B preceding e lies on q (and not on p). Let $\prec \subseteq \mathcal{P} \times \mathcal{P}$ denote this relation.*

A $\prec$-minimal path ending in a given reachable node w can be computed by a simple backward-search on the sub-graph of reachable nodes prioritizing non-boarding edges over other edges. This in fact returns a $\prec$-minimal path as for any conflicting edge e with a path $q \in \mathcal{P}$, the corresponding boarding edge e_B must lie on q. As the graph is acyclic, this backward-search terminates in $\mathcal{O}(|E|)$ time.

▶ **Lemma 11.** *For the end node w of any path in $\mathcal{P}$, a $\prec$-minimal path ending in w can be computed in $\mathcal{O}(|E|)$ time.*

In order to compute single-commodity equilibrium flows, we can now successively send flow along $\prec$-minimal and τ-optimal paths. In every iteration, the flow on this path is increased until an edge becomes fully saturated. Then, we reduce the capacity on the edges of p by the added flow, remove zero-capacity edges, and repeat this procedure until the demand is met.

▶ **Theorem 12.** *For single-commodity networks, a user equilibrium can be computed in $\mathcal{O}(|E|^2)$ time. The resulting user equilibrium uses at most $|E|$ paths.*

For general (aperiodic) schedules our algorithm is strongly polynomial in the input. For compactly describable periodic schedules it is only pseudo-polynomial as it depends on the size of the time-expanded network. The actual blow-up of the network depends on the ratio of the time horizon and the period length.

4.2 The General Multi-Commodity Case

The approach of the previous section fails for the general multi-commodity case as the set of paths $\bigcup_i \mathcal{P}_i$ may not necessarily have a $\prec$-minimal element if there are commodities that do not share the same destination station. Hence, we now describe a finite-time algorithm and a heuristic for computing multi-commodity equilibria in practice.

4.2.1 A finite-time algorithm

In the following, we describe a finite-time algorithm for computing exact multi-commodity user equilibria. As we know that an equilibrium f exists, the idea is now to guess the subset E_O of driving edges that are at capacity, i.e., $E_O = \{e \in E_D \mid f_e = \nu_e\}$, as well as a positive, lower bound ε on the available capacity on all other edges, i.e., $\min_{e \in E_D \setminus E_O} \nu_e - f_e \geq \varepsilon$. Then, we check the following set defined by linear constraints for feasibility:

$$\mathcal{F}(E_O, \varepsilon) := \left\{ f \in \mathcal{F}_Q \;\middle|\; \begin{array}{rl} f_e = \nu_e, & \text{for } e \in E_O, \\ f_e \leq \nu_e - \varepsilon, & \text{for } e \in E_D \setminus E_O, \\ f_{i,p} = 0, & \text{for } i \in I,\, p \in \mathcal{P}_i(E_O) \end{array} \right\}, \tag{5}$$

where $\mathcal{P}_i(E_O)$ is the set of paths $p \in \mathcal{P}_i$ for which there exists a $q \in \mathcal{P}_i$ with $\tau_q < \tau_p$ such that $e^+ \notin E_O$ holds for all edges $e \in E_B \cap q$ with $e^+ \notin p$.

▶ **Lemma 13.** *The set of user equilibria coincides with* $\bigcup_{\varepsilon>0}\bigcup_{E_O\subseteq E_D}\mathcal{F}(E_O,\varepsilon)$.

As the set $\mathcal{F}(E_O,\varepsilon)$ only grows when reducing ε, a finite-time algorithm can iteratively decrease ε and then check the feasibility of $\mathcal{F}(E_O,\varepsilon)$ for every subset E_O of E_D. As a user equilibrium must exist, an equilibrium will be found for small enough ε.

▶ **Corollary 14.** *The procedure described above computes a user equilibrium in finite time.*

4.2.2 Heuristic for computing multi-commodity equilibria

In the following, we describe a heuristic for computing multi-commodity equilibria. Start with some initial feasible flow $f \in \mathcal{F}_Q^\nu$, e.g., by sending all flow along their outside option. Then, iteratively, find a direction $d \in \mathbb{R}^\mathcal{P}$ and change the flow along this direction while preserving feasibility, until an equilibrium is found. More specifically, we replace f with $f' = f + \alpha \cdot d$ where α is maximal such that f' is feasible.

▶ **Definition 15.** *Let f be a feasible flow. A* direction $d \in \mathbb{R}^\mathcal{P}$ *is called*

- balanced *if* $\sum_{p\in\mathcal{P}_i} d_{i,p} = 0$ *for* $i \in I$*, and*
- feasible for f *if the flow* $f + \alpha \cdot d$ *is feasible for small enough* $\alpha > 0$.

Clearly, the choice of the direction is essential for this heuristic to approach an equilibrium. The characterization in Theorem 2 indicates using a direction d such that $f + \alpha \cdot d$ is a deviation violating the quasi-variational inequality, i.e., $d := 1_{i,q} - 1_{i,p}$ for some path $p \in \mathcal{P}_i$ with $f_{i,p} > 0$ for which q is a better available alternative, i.e., $q \in A_{i,p}(f)$ and $\tau_q < \tau_p$. However, not all such directions are feasible; even worse, sometimes no feasible direction is of this form. Therefore, our approach is to start with such a direction d and, if necessary, transform it to make it feasible.

If this heuristic terminates, it provides an equilibrium, but termination is not always guaranteed, as we will see later. We first describe how we achieve feasibility of the direction. For this, a key observation is stated in the following proposition:

▶ **Proposition 16.** *Let f be a feasible flow and d a balanced direction that fulfils $f_{i,p} > 0$ whenever $d_{i,p} < 0$. Then, d is a feasible direction for f if and only if there exists no boarding edge e such that $f_{e^+} = \nu_{e^+}$, $d_{e^+} > 0$ and ($f_e > 0$ or $d_e > 0$) hold.*

In the case that $d = 1_{i,q} - 1_{i,p}$ is an infeasible direction, we apply the following transformation: As long as d is infeasible, there exists a boarding edge such that $f_{e^+} = \nu_{e^+}$, $d_{e^+} > 0$, and $f_e > 0 \vee d_e > 0$, and we repeat the following procedure: Let (i, p') be such that p' is a path containing e with positive flow $f_{i,p'} > 0$ or whose entry in the direction vector is positive, i.e., $d_{i,p'} > 0$. We decrease $d_{i,p'}$ by $\delta := d_e$, if $f_{i,p'} > 0$, or by $\delta := \min(d_e, d_{i,p'})$, otherwise. Next, we determine a best path $q' \in \mathcal{P}_i$ that does not use full driving edges, i.e., driving edges $\tilde{e}$ with $f_{\tilde{e}} = \nu_{\tilde{e}}$ and $d_{\tilde{e}} \geq 0$. We increase $d_{i,q'}$ by $\min(\{\delta\} \cup \{-d_e \mid e \in q', f_e = \nu_e\})$, and afterwards, decrease δ by the same amount. We repeat this until δ equals zero. A detailed description of this transformation of the direction can be found in Algorithm 1.

▶ **Proposition 17.** *Algorithm 1 transforms any direction d, that fulfils $f_{i,p} > 0$ whenever $d_{i,p} < 0$, to a feasible direction.*

Using Algorithm 1 in the main loop, we can implement the heuristic mentioned above. However, in some situations, the heuristic might apply changes along directions $d_1, \ldots, d_k$ in a cyclic behaviour. We distinguish between *terminating* cycles, for which the heuristic breaks out of the cyclic behaviour after some finite but potentially very large number of

Algorithm 1 Establishing feasible directions.

```
Data: Time-expanded graph with outside options, feasible flow $f$, balanced
      direction $d \in \mathbb{Z}^{\mathcal{P}}$ s.t. $\forall(i,p) : d_{i,p} < 0 \implies f_{i,p} > 0$
Result: A feasible direction
while $\exists e \in E_B$ with $f_{e^+} = \nu_{e^+} \wedge d_{e^+} > 0 \wedge (f_e > 0 \vee d_e > 0)$ do
    $(i,p) \leftarrow$ any commodity $i$ and path $p$ containing $e$ with $f_{i,p} > 0$ or $d_{i,p} > 0$;
    $\delta \leftarrow \begin{cases} d_{e^+}, & \text{if } f_{i,p} > 0, \\ \min(d_{e^+}, d_{i,p}), & \text{otherwise.} \end{cases}$;
    Decrease $d_{i,p}$ by $\delta$;
    while $\delta > 0$ do
        $q \leftarrow$ best alternative to $p$ not containing any $e' \in E_D$ with $f_{e'} = \nu_{e'} \wedge d_{e'} \geq 0$;
        $\delta' \leftarrow \min(\{\delta\} \cup \{-d_{e'} \mid e' \in q, f_{e'} = \nu_{e'}\})$;
        Increase $d_{i,q}$ by $\delta'$;
        Decrease $\delta$ by $\delta'$;
return $d$
```

iterations, and *non-terminating* cycles. In practice, most terminating cycles can be detected and prohibited by changing the flow along the common direction $\sum_{i=1}^{k} d_i$. Non-terminating cycles, however, constitute a more serious problem. We can detect these cycles, as their common direction $\sum_{i=1}^{k} d_i$ vanishes. Randomizing the path selection in the main loop of the heuristic might help in breaking out of the cycle.

We employ a technique to reduce the initial complexity of a given instance: There is a class of paths for which the procedure will never remove flow from. Thus, we first fill these paths directly when initializing the heuristic. In our experiments, this has shown to handle between 10% and 25% of the total demand before entering the heuristic.

5 Computational Study

To gain insights into the applicability of the proposed heuristic, we conduct a computational study on real world train networks. We analyse the performance of the heuristic and compare the computed equilibrium solutions with system optima.

5.1 Experiment Setup

For each network in our dataset, we use a dynamic demand profile. As our dataset only provides aggregate, static demand data, we generate a dynamic profile using a uniform distribution of the demand over time horizon of a typical work day. This means, for every origin-destination (OD) pair, we generate commodities c_i with varying start time.

Since the performance of our heuristic varies with the utilization of network capacities, we also apply it to rescaled demands. A demand scale factor of 1 reflects the real-world demand.

The system optima are computed using a linear-programming based approach with delayed column generation. A system optimum is an optimal solution to the linear program

$$\min_{f \in \mathcal{F}_Q^{\nu}} \sum_{(i,p) \in \mathcal{P}} \tau_p \cdot f_{i,p}.$$

This problem is solved using a column generation approach. It is initialized with some feasible solution, e.g., by sending all particles along their outside option. Then, after every

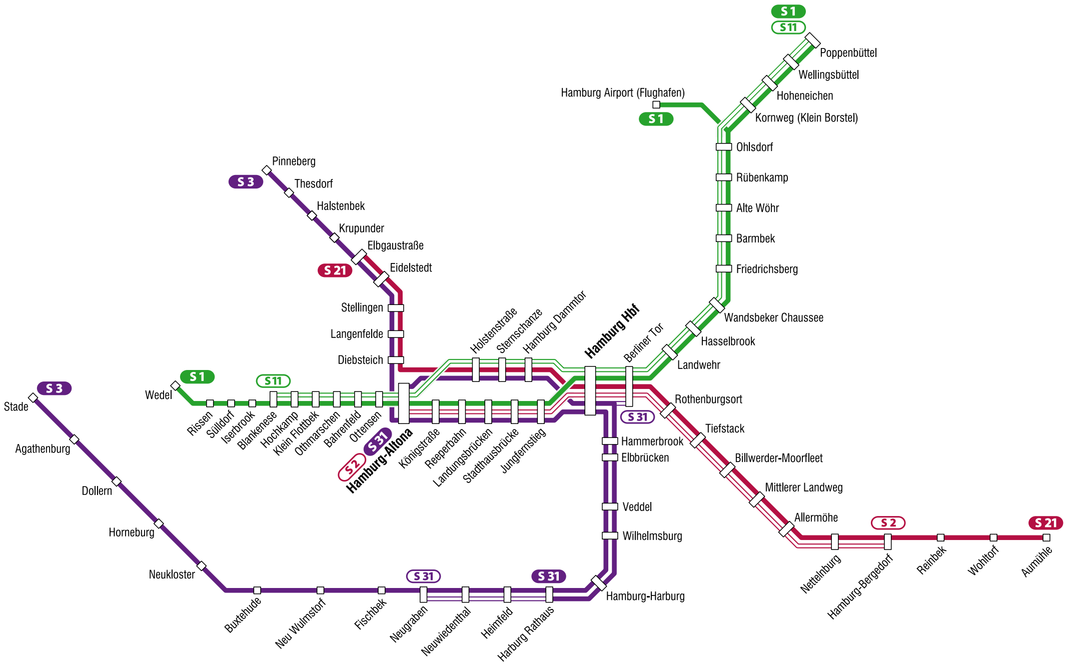


Figure 2 Map of the considered S-Bahn Hamburg as operated until 2023 [1].

iteration, paths that reduce the objective function are added to the so-called Master-LP, and if any path was added, the Master-LP is solved again. For this, we used software developed by [21, 22].

Our implementation of the heuristic (available in [20]) uses the Rust programming language. All experiments were conducted on an AMD Ryzen 9 5950X CPU.

5.2 Data

A pool of periodic timetables of real-world public transportation networks is provided in the publicly available TimPassLib [35]. In our computational study, we unroll these periodic timetables to match an observation period of a day (18 hours) and feed our heuristic the resulting schedule. Table 1 describes the considered networks and schedules in more detail.

We present results for the Hamburg S-Bahn network (Figure 2) and the Swiss long-distance train network. The nominal demand of 750.000 passengers per day in the Hamburg S-Bahn network was taken from [34]. Due to a lack of real-world data, we set the capacity of all vehicles to 1000 passengers, a number closely aligned to real-world train systems. For example, a 2-unit train system of the DBAG Class 490 can hold up to 1028 passengers. The average number of stops per vehicle is 19.14 in the Hamburg network and 8.25 in the Swiss network. We equip all commodities in our experiments with an outside option of 180 minutes.

Table 1 Details of the considered networks.

Name	# stations	# vehicles	nominal demand	# commodities
Hamburg S-Bahn	68	1,512	750,000	219,240
Swiss Long Distance	140	2,772	1,347,686	2,609,712

5.3 Results

For real-world demands, the heuristic computes user equilibria in both networks in less than 30 seconds. Starting with a demand factor of 0.5 for the Hamburg S-Bahn network and 0.55 for the Swiss network, capacity conflicts occur; for demand factors below these values, any system-optimal flow is also an equilibrium flow as all agents can be assigned to a path p minimizing τ_p over all paths $p \in \mathcal{P}_i$. With an increase in demand, the number of iterations of the heuristic increases heavily. For example in the Hamburg instance, when initializing the heuristic by filling optimal paths until all of them have a full driving edge, only 1,015 iterations are necessary for the nominal demand while 10,280,576 iterations were necessary for a demand factor of 2.5. For the Swiss long-distance train network, the computation time increased similarly dramatically already when doubling the demand.

The average travel time in the computed equilibria is up to 3.3% higher, in the Hamburg instance, and up to 7.4% higher, in the Swiss instance, compared to a system optimum. For the nominal demand, these numbers are below 1%. Figure 3 shows the computation time of the heuristic, and the mean travel times in the equilibrium and in the system optimum.

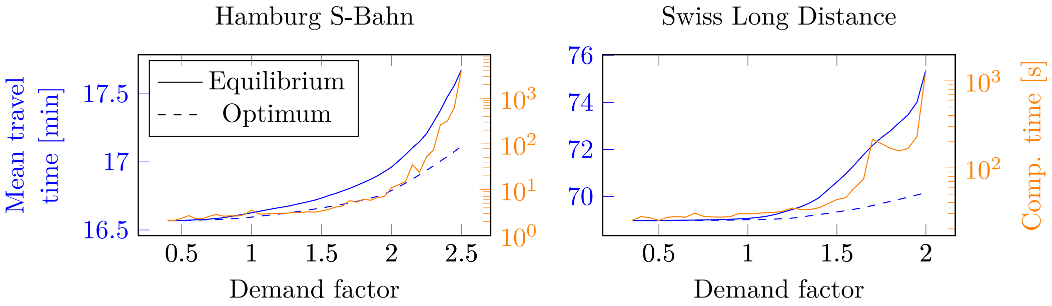


Figure 3 Comparison of the mean travel time of the user equilibrium and system optimum, and the computation time of the heuristic in the two networks.

6 Conclusion

We presented a side-constrained user equilibrium model for a schedule-based transit network incorporating hard vehicle capacities. As our main results, we proved that equilibria exist and can be computed efficiently for single-commodity instances. The existence result generalizes a classical result of Bernstein and Smith [2]; its proof is based on a new condition (weak regularity) implying existence of BS-equilibria for a class of discontinuous and non-separable cost maps. For general multi-commodity instances we devised a heuristic, which was implemented and tested on several realistic networks based on data of the Hamburg S-Bahn and the Swiss railway.

Limitations of the Model

Our model assumes that passengers are associated with fixed start times for their travel. For flexible departure time choices, however, a side-constrained user equilibrium need not exist (see [28]). Non-existence also applies to a model with transfer penalties.

Open Problems

First of all, a side-constrained user equilibrium is not unique and, hence, the issue of equilibrium selection or determining which equilibrium is likely to be observed in practice remains unclear and deserves further study. From an algorithmic point of view, the hardness of the equilibrium computation problem for multi-commodity networks is open. The computational complexity for single-commodity networks and periodic timetables (which leads to a compactly representable time-expanded graph) is also open.

References

1 Arbalete. Liniennetzplan der S-Bahn Hamburg, 2016. Licensed under CC BY-SA 4.0. Accessed 19-Dec-2023. URL: `https://commons.wikimedia.org/wiki/File:Karte_der_S-Bahn_Hamburg.svg`.

2 David Bernstein and Tony E. Smith. Equilibria for networks with lower semicontinuous costs: With an application to congestion pricing. *Transp. Sci.*, 28(3):221–235, August 1994. doi:10.1287/trsc.28.3.221.

3 Belgacem Bouzaïene-Ayari, Michel Gendreau, and Sang Nguyen. Passenger assignment in congested transit networks: A historical perspective. In Patrice Marcotte and Sang Nguyen, editors, *Equilibrium and Advanced Transportation Modelling*, pages 47–71. Springer, Boston, MA, 1998. doi:10.1007/978-1-4615-5757-9_3.

4 Paolo Carraresi, Federico Malucelli, and Stefano Pallottino. Regional mass transit assignment with resource constraints. *Transp. Res. B: Methodol.*, 30(2):81–98, April 1996. doi:10.1016/0191-2615(95)00027-5.

5 M. Cepeda, R. Cominetti, and M. Florian. A frequency-based assignment model for congested transit networks with strict capacity constraints: characterization and computation of equilibria. *Transp. Res. B: Methodol.*, 40(6):437–459, July 2006. doi:10.1016/j.trb.2005.05.006.

6 Roberto Cominetti and José R. Correa. Common-lines and passenger assignment in congested transit networks. *Transp. Sci.*, 35(3):250–267, 2001. doi:10.1287/trsc.35.3.250.10154.

7 José R. Correa, Andreas S. Schulz, and Nicolás E. Stier-Moses. Selfish routing in capacitated networks. *Math. Oper. Res.*, 29(4):961–976, November 2004. doi:10.1287/moor.1040.0098.

8 José R. Correa and Nicolás E. Stier-Moses. Wardrop equilibria. In James J. Cochran, Louis A. Cox, Jr, Pinar Keskinocak, Jeffrey P. Kharoufeh, and J. Cole Smith, editors, *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Ltd, 2011. doi:10.1002/9780470400531.eorms0962.

9 Umberto Crisalli. Dynamic transit assignment algorithms for urban congested networks. *WIT Trans. Built Environ.*, 44, 1999. URL: `https://www.witpress.com/elibrary/wit-transactions-on-the-built-environment/44/5985`.

10 Stella C. Dafermos and Frederick T. Sparrow. The traffic assignment problem for a general network. *J. Res. Natl. Inst. Stand.*, 73B(2):91–118, April 1969. doi:10.6028/jres.073B.010.

11 Joaquín de Cea and Enrique Fernández. Transit assignment for congested public transport systems: An equilibrium model. *Transp. Sci.*, 27(2):133–147, 1993. doi:10.1287/trsc.27.2.133.

12 Qian Fu, Ronghui Liu, and Stephane Hess. A review on transit assignment modelling approaches to congested networks: a new perspective. *Procedia Soc. Behav. Sci.*, 54:1145–1155, October 2012. doi:10.1016/j.sbspro.2012.09.829.

13 Guido Gentile, Michael Florian, Younes Hamdouch, Oded Cats, and Agostino Nuzzolo. The theory of transit assignment: Basic modelling frameworks. In Guido Gentile and Klaus Noekel, editors, *Modelling Public Transport Passenger Flows in the Era of Intelligent Transport Systems*, STTT, pages 287–386. Springer, Cham, 2016. doi:10.1007/978-3-319-25082-3_6.

14 Lukas Graf and Tobias Harks. Side-constrained dynamic traffic equilibria. In Kevin Leyton-Brown, Jason D. Hartline, and Larry Samuelson, editors, *Proceedings of the 24th ACM Conference on Economics and Computation*, page 814. ACM, 2023. doi:10.1145/3580507.3597801.

15 Younes Hamdouch, H.W. Ho, Agachai Sumalee, and Guodong Wang. Schedule-based transit assignment model with vehicle capacity and seat availability. *Transp. Res. B: Methodol.*, 45(10):1805–1830, December 2011. doi:10.1016/j.trb.2011.07.010.

16 Younes Hamdouch and Siriphong Lawphongpanich. Schedule-based transit assignment model with travel strategies and capacity constraints. *Transp. Res. B: Methodol.*, 42(7-8):663–684, August 2008. doi:10.1016/j.trb.2007.11.005.

17 Younes Hamdouch, Patrice Marcotte, and Sang Nguyen. Capacitated transit assignment with loading priorities. *Math. Program.*, 101(1):205–230, 2004. doi:10.1007/s10107-004-0542-7.

18 Younes Hamdouch, W. Y. Szeto, and Y. Jiang. A new schedule-based transit assignment model with travel strategies and supply uncertainties. *Transp. Res. B: Methodol.*, 67:35–67, September 2014. doi:10.1016/j.trb.2014.05.002.

19 Tobias Harks, Sven Jäger, Michael Markl, and Philine Schiewe. Computing user equilibria for schedule-based transit networks with hard vehicle capacities, 2024. doi:10.48550/arXiv.2406.17153.

20 Tobias Harks, Sven Jäger, Michael Markl, and Philine Schiewe. A heuristic for computing user equilibria in schedule-based transit networks with hard capacity constraints, 2024. Software, swhId: swh:1:dir:a3e48de8912d81166b2517c3c826e11bcda2c356 (visited on 2024-08-20). URL: https://github.com/ArbeitsgruppeTobiasHarks/sbta/tree/atmos.

21 Tobias Harks, Ingo Kleinert, Max Klimm, and Rolf H. Möhring. Computing network tolls with support constraints. *Networks*, 65(3):262–285, March 2015. doi:10.1002/net.21604.

22 Olaf Jahn, Rolf H. Möhring, Andreas S. Schulz, and Nicolás E. Stier-Moses. System-optimal routing of traffic flows with user constraints in networks with congestion. *Oper. Res.*, 53(4):600–616, August 2005. doi:10.1287/opre.1040.0197.

23 Fumitaka Kurauchi, Michael G. H. Bell, and Jan-Dirk Schmöcker. Capacity constrained transit assignment with common lines. *J. Math. Model. Algorithms*, 2(4):309–327, December 2003. doi:10.1023/b:jmma.0000020426.22501.c1.

24 Homero Larrain, Hemant K. Suman, and Juan Carlos Muñoz. Route based equilibrium assignment in congested transit networks. *Transp. Res. C: Emerg. Technol.*, 127:103125, June 2021. doi:10.1016/j.trc.2021.103125.

25 Torbjörn Larsson and Michael Patriksson. An augmented lagrangean dual algorithm for link capacity side constrained traffic assignment problems. *Transp. Res. B: Methodol.*, 29(6):433–455, December 1995. doi:10.1016/0191-2615(95)00016-7.

26 Patrice Marcotte and Sang Nguyen. Hyperpath formulations of traffic assignment problems. In Patrice Marcotte and Sang Nguyen, editors, *Equilibrium and Advanced Transportation Modelling*, pages 175–200. Springer, Boston, MA, 1998. doi:10.1007/978-1-4615-5757-9_9.

27 Patrice Marcotte, Sang Nguyen, and Alexandre Schoeb. A strategic flow model of traffic assignment in static capacitated networks. *Oper. Res.*, 52(2):191–212, April 2004. doi:10.1287/opre.1030.0091.

28 Sang Nguyen, Stefano Pallottino, and Federico Malucelli. A modeling framework for passenger assignment on a transport network with timetables. *Transp. Sci.*, 35(3):238–249, August 2001. doi:10.1287/trsc.35.3.238.10152.

29 Agostino Nuzzolo and Francesco Russo. Stochastic assignment models for transit low frequency services: Some theoretical and operative aspects. In Lucio Bianco and Paolo Toth, editors, *Advanced Methods in Transportation Analysis*, Transportation Analysis, pages 321–339. Springer, 1996. doi:10.1007/978-3-642-85256-5_14.

30 Agostino Nuzzolo, Francesco Russo, and Umberto Crisalli. A doubly dynamic schedule-based assignment model for transit networks. *Transp. Sci.*, 35(3):268–285, August 2001. doi:10.1287/trsc.35.3.268.10149.

31 Natale Papola, Francesco Filippi, Guido Gentile, and Lorenzo Meschini. Schedule-based transit assignment: new dynamic equilibrium model with vehicle capacity constraints. In Agostino Nuzzolo and Nigel H. M. Wilson, editors, *Schedule-Based Modeling of Transportation Networks*, volume 46 of *ORCS*, pages 1–26. Springer, 2009. doi:10.1007/978-0-387-84812-9_8.

32 Michael Patriksson. *The traffic assignment problem: models and methods*. VSP, Utrecht, 1994.

33 Normen Rochau, Klaus Nökel, and Michael G. H. Bell. Cost functions for strategies in schedule-based transit networks with limited vehicle capacities. *Transp. Res. Rec.*, 2284(1):62–69, January 2012. doi:10.3141/2284-08.

34 S-Bahn Hamburg GmbH. Zahlen, Daten, Fakten rund um die S-Bahn Hamburg, 2023. URL: https://www.s-bahn-hamburg.de/service/schulklassen/daten-zahlen-fakten.

35 Philine Schiewe, Marc Goerigk, and Niels Lindner. Introducing TimPassLib – A library for integrated periodic timetabling and passenger routing. *Oper. Res. Forum*, 4, August 2023. doi:10.1007/S43069-023-00244-1.

36 Mike J. Smith. The existence, uniqueness and stability of traffic equilibria. *Transp. Res. B: Methodol.*, 13(4):295–304, December 1979. doi:10.1016/0191-2615(79)90022-5.

37 Mike J. Smith. Two alternative definitions of traffic equilibrium. *Transp. Res. B: Methodol.*, 18(1):63–65, February 1984. doi:10.1016/0191-2615(84)90006-7.

38 Heinz Spiess and Michael Florian. Optimal strategies: A new assignment model for transit networks. *Transp. Res. B: Methodol.*, 23(2):83–102, April 1989. doi:10.1016/0191-2615(89)90034-9.

39 John Glen Wardrop. Some theoretical aspects of road traffic research. *Proc. Inst. Civ. Eng.*, 1(3):325–378, May 1952.

40 Jia Hao Wu, Michael Florian, and Patrice Marcotte. Transit equilibrium assignment: A model and solution algorithms. *Transp. Sci.*, 28(3):193–203, August 1994. doi:10.1287/trsc.28.3.193.

41 Z. X. Wu and William H. K. Lam. Network equilibrium for congested multi-mode networks with elastic demand. *J. Adv. Transp.*, 37(3):295–318, 2003. doi:10.1002/atr.5670370304.

Dynamic Traffic Assignment for Public Transport with Vehicle Capacities

Julian Patzner
Martin-Luther-Universität Halle-Wittenberg, Germany

Matthias Müller-Hannemann
Martin-Luther-Universität Halle-Wittenberg, Germany

Abstract

Traffic assignment is a core component of many urban transport planning tools. It is used to determine how traffic is distributed over a transportation network. We study the task of computing traffic assignments for public transport: Given a public transit network, a timetable, vehicle capacities and a demand (i.e. a list of passengers, each with an associated origin, destination, and departure time), the goal is to predict the resulting passenger flow and the corresponding load of each vehicle. Microscopic stochastic simulation of individual passengers is a standard, but computationally expensive approach. Briem et al. (2017) have shown that a clever adaptation of the Connection Scan Algorithm (CSA) can lead to highly efficient traffic assignment algorithms, but ignores vehicle capacities, resulting in overcrowded vehicles. Taking their work as a starting point, we here propose a new and extended model that guarantees capacity-feasible assignments and incorporates dynamic network congestion effects such as crowded vehicles, denied boarding, and dwell time delays. Moreover, we also incorporate learning and adaptation of individual passengers based on their experience with the network. Applications include studying the evolution of perceived travel times as a result of adaptation, the impact of an increase in capacity, or network effects due to changes in the timetable such as the addition or the removal of a service or a whole line. The proposed framework has been experimentally evaluated with public transport networks of Göttingen and Stuttgart (Germany). The simulation proves to be highly efficient. On a standard PC the computation of a traffic assignment takes just a few seconds per simulation day.

2012 ACM Subject Classification Applied computing → Transportation

Keywords and phrases Public transport, traffic assignment, vehicle capacities, crowding, stochastic simulation, learning

Digital Object Identifier 10.4230/OASIcs.ATMOS.2024.18

Related Version *Full Version*: https://arxiv.org/abs/2408.06308

1 Introduction

Efficient, sustainable, and accessible public transport systems are critical to promoting economic growth, reducing congestion and minimizing environmental impact. This calls for innovative methods to optimize resource allocation, improve passenger comfort and ensure the overall efficiency of transit networks. A crucial part in the planning process of public transit systems is traffic assignment. Traffic assignment models are used to predict the passenger flow and the estimated load of vehicles within a transit network for a given demand scenario, making them a fundamental analysis and evaluation tool at both planning and operational levels [4]. Results of traffic assignments provide valuable insights into possible congestion problems due to insufficient capacity. They can be used to study the benefits of introducing additional services, increased frequencies, larger vehicle capacities or possible network extensions [5]. In this work, we consider the following variant of public traffic assignment: As input we are given a public transit network, a corresponding timetable and a

24th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2024).
Editors: Paul C. Bouman and Spyros C. Kontogiannis; Article No. 18; pp. 18:1–18:20
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

vehicle schedule with vehicle capacities. The demand is specified by a list of passengers, each with an associated origin, destination, and departure time. The task is to assign for each individual passenger a journey from his origin to his destination.

Microscopic stochastic simulation of individual passengers is meanwhile a standard, but computationally expensive approach. Briem et al. [1] have shown that a clever adaptation of the Connection Scan Algorithm (CSA) [7] can lead to highly efficient traffic assignment algorithms. However, their approach ignores vehicle capacities, resulting in unrealistic assignments and overcrowded vehicles (they report in their case study assignments of about 1200 passengers to a single vehicle). The commercial state-of-the-art tool PTV VISUM has recently integrated CSA into their transport assignment for faster shortest path search [12].

Contribution. Taking the work of Briem et al. [1] as a starting point, we here propose a new and extended model that guarantees capacity-feasible assignments. We use agent-based modeling, a powerful tool to study the behavior of passengers, transport vehicles and the interaction between them. By modeling passengers as autonomous agents, this approach captures the different decision-making processes, preferences and adaptive behaviors that individuals exhibit during their journeys. More specifically, our model incorporates dynamic network congestion effects such as crowded vehicles, denied boarding, and dwell time delays. Moreover, we also incorporate learning and adaptation of individual passengers based on their experience with the network. The proposed model has been implemented as a prototype. Computational experiments with public transport networks of Göttingen and Stuttgart (Germany) demonstrate the efficiency of the approach. We present three case studies with selected applications:

1. First, we study how passengers respond to network congestion. We find that the learning process is quite effective. It helps to improve the average perceived travel times and to reduce cases of denied boardings due to overcrowded vehicles.
2. Second, we examine the benefits of increasing capacity. It turns out that a moderate increase in capacity leads to a significant reduction in average perceived travel times.
3. Third, we compare unlimited vs. limited vehicle capacity. As expected the passenger flows with unlimited vehicle capacity turn out to be highly unrealistic.

Related work. There is a long history of research on traffic assignment in public transport, see [9, 10, 11] for surveys. Conventional traffic assignment models distinguish between frequency-based and timetable-based models. These two groups differ in the modeling of the network. In frequency-based models [16, 20, 25, 24] the timetable is only modeled at the line level. Each line has an assigned frequency. These models aim at determining the average loads on the lines. In timetable-based models [13, 14, 17, 18, 19], the trips of a line are explicitly modeled, and the task is to determine loads on each single trip. A prominent example of the implementation of a schedule-based model is the commercial software VISUM, which is primarily used for long-term planning. In agent-based models, passengers are not considered as an aggregated flow, but are simulated individually on a microscopic level. The individual vehicles are also modeled as individual agents, which allows great freedom in modeling (for example, the development of vehicle-specific delays or seating and standing capacities). Agent-based models focus on the dynamic interactions between passengers and the network as well as interactions between passengers. Individual, adaptive decisions are simulated as a reaction to dynamic network conditions. The network conditions are in turn dependent on the individual decisions of the passengers. In addition to dynamic processes within a day, a learning process lasting several days is usually modeled. The experiences on one day are incorporated into the expectations of the individual passengers and thus influence the decisions on subsequent days. These learning processes model long-term adaptations of passengers to the network conditions. In 2008, Wahba presented MILATRAS [26, 27], the first agent-based simulation in public transport that models a learning process. In MILATRAS, the traffic assignment is considered as a Markov decision problem, where the possible positions of the passengers (stops, vehicles) are the states and the possible decisions (choice of the next line or stop to alight) are the actions. In 2009, MATSim, an activity-based agent simulation framework, was extended by Rieser *et al.* to include public transportation trips [21]. In MATSim, each traveler has a population of plans representing journeys. Each passenger randomly selects a plan from its population. This selection is based on journey ratings and the learning process is implemented as a co-evolutionary algorithm.

With BusMezzo [2, 4, 5] another agent-based simulation was introduced by Cats in 2011, designed as an operations-oriented model for short-term to mid-term planning [3]. The probabilistic decisions in BusMezzo depend on the current expectations of passengers, based on previous days' experiences and current real-time information. The individual decisions (boarding, alighting and walking) depend on pre-computed path sets, where each action (e.g. alighting at a specific stop or boarding a specific vehicle) is assigned a path set (e.g. a subset of all possible paths to the destination after alighting). In [4] a path is defined as a sequence of stops, whereby the exact lines and transfers are not specified. A single path is therefore not a concrete journey. The expected waiting time at a stop is calculated based on the combined frequency of the lines at the stop. A SoftMax model is used when deciding between different actions. Passengers learn the perceived travel times and the waiting times for the individual path segments. In contrast to MILATRAS and BusMezzo, the model proposed in this paper avoids the static pre-computation of alternative path sets. Instead, we consider and evaluate all feasible actions dynamically on-the-fly in an event-based manner, allowing passengers to react in a flexible way on network conditions such as unexpected delays or congestion. In our model the evaluation of individual passenger decisions depends on explicit journeys, which include specifically defined trips and transfers. Due to the explicit definition of journeys, the model is also suitable for timetables that include routes with low frequencies or individual special trips. Passengers can learn the expected load and reliability of specific trips, not only about lines. Similarly, probabilities of failed transfers can be learned.

Overview. The remainder of this paper is structured as follows. First, we start with the necessary preliminaries to formalize the problem in Section 2. In Section 3, we introduce our framework in detail. In Section 4, we present a computational study evaluating our framework and showcasing a few applications. Finally, we conclude with a short summary.

2 Preliminaries

This section describes the modeling of the network and provides basic definitions and notations. A timetable is modeled as an event-activity network [15]. An *event-activity-network* is a tuple $(\mathcal{E}, \mathcal{A}, \mathcal{S}, \mathcal{T}, \mathcal{L}, \mathcal{F}, \mathcal{D})$ whose components are described below. The *events* $\mathcal{E}$ and the *activities* $\mathcal{A}$ form a network $N = (\mathcal{E}, \mathcal{A})$, where the events correspond to the nodes and the activities to the arcs. The events are divided into *departure events* $\mathcal{E}_{dep}$ and *arrival events* $\mathcal{E}_{arr}$. Each event e is associated with a time $\tau(e)$, a *trip* $trip(e) \in \mathcal{T}$, and a *stop* $stop(e) \in \mathcal{S}$. We write $dep(s)$ and $arr(s)$ for the set of all departure and arrival events at stop s. An *activity* (e_1, e_2) can be a driving, dwelling or transfer activity. A *dwelling arc* is an arc from an arrival event to a departure event, modeling the waiting of a vehicle at a stop. *Driving* *arcs* are arcs from a departure to an arrival event and model driving from one stop to the next. Driving and dwelling activities have an *in-vehicle time* $\tau^{ivt}(e_1, e_2) := \tau(e_2) - \tau(e_1)$ and a *minimum in-vehicle time* $\tau^{ivt}_{min}(e_1, e_2)$. The difference between the regular duration and the minimum duration of an arc corresponds to the catch-up potential in case of delays. A driving or dwelling activity (e_1, e_2) has a reference to its trip $\mathit{trip}(e_1, e_2) \in \mathcal{T}$.

A *trip* $t \in \mathcal{T}$ is an alternating sequence of departure and arrival events $(e^1_{dep}(t), e^2_{arr}(t), e^2_{dep}(t), ..., e^{|S(t)|-1}_{arr}(t), e^{|S(t)|-1}_{dep}(t), e^{|S(t)|}_{arr}(t))$, where $S(t)$ is the set of stops served by the trip t. Denote by $e^i_{dep}(t)$ the departure event and by $e^i_{arr}(t)$ the arrival event at the ith stop of t. The times of the events of a trip are non-decreasing, so $\tau(e^i_{dep}(t)) \leq \tau(e^{i+1}_{arr}(t))$ and $\tau(e^i_{arr}(t)) \leq \tau(e^i_{dep}(t))$ always apply. This sequence of events defines the alternating sequence of driving and dwelling *activities*(t) of trip t. For a *trip segment* between the ith and jth stop of a trip t, with $i < j$, we write $e^i_{dep}(t) \to e^j_{arr}(t)$. This trip segment contains all driving and dwelling arcs between the departure event at the ith stop and the arrival event at the jth stop of trip t. Let $\mathit{activities}(e^i_{dep}(t) \to e^j_{arr}(t))$ be this sequence of arcs. Each trip has a *seat capacity* $\mathit{cap}_{sit}(t)$, which corresponds to the number of seats in the vehicle. The *capacity* $\mathit{cap}(t) \geq \mathit{cap}_{sit}(t)$ of a trip is the sum of the seats and standing capacity. This capacity is considered as a hard upper limit for the number of passengers that can be on a trip. Trips are grouped into *lines* $\mathcal{L}$, where each trip of a line serves the same sequence of stops. Let $\mathit{line}(t)$ be the line of a trip t and $\mathit{line}(e)$ the line of an event e. We assume that two trips of a line cannot overtake each other. A line is therefore a set of trips ordered according to the first departure time. Let t_1 and t_2 be two subsequent trips of a line. The *headway* $\mathit{headway}(e^i_x(t_1)) := \tau(e^i_x(t_2)) - \tau(e^i_x(t_1))$ of an event is the time until the corresponding event of the next trip. A *transfer* is an arc from an arrival event to a departure event of another trip. Such arcs are not explicitly modeled, but are implicitly defined by the stops $\mathcal{S}$ and *footpaths* $\mathcal{F}$. A footpath $(s, s') \in \mathcal{F}$ between two stops can be passed at any time. The time required for a footpath is given by $\ell(s, s') \in \mathbb{N}$. A *minimum transfer time* $\mathit{mct}(s)$ can be specified for transferring at stop s. A transfer $e_{arr} \to e_{dep}$ with $\mathit{trip}(e_{arr}) \neq \mathit{trip}(e_{dep})$ is therefore *valid* if either $\mathit{stop}(e_{arr}) = \mathit{stop}(e_{dep})$ and $\tau(e_{arr}) + \mathit{mct}(\mathit{stop}(e_{arr})) \leq \tau(e_{dep})$ applies, or if $\mathit{stop}(e_{arr}) \neq \mathit{stop}(e_{dep})$ and the footpath $(\mathit{stop}(e_{arr}), \mathit{stop}(e_{dep}))$ exists with $\tau(e_{arr}) + \ell(\mathit{stop}(e_{arr}), \mathit{stop}(e_{dep})) \leq \tau(e_{dep})$. A valid transfer can become invalid in the course of the simulation due to delays. Conversely, an invalid transfer can also become valid if the departure event is delayed. The *walking time* $\tau^{walk}(e_{arr} \to e_{dep})$ of a transfer is $\ell(\mathit{stop}(e_{arr}), \mathit{stop}(e_{dep}))$ if $\mathit{stop}(e_{arr}) \neq \mathit{stop}(e_{dep})$, and 0 otherwise. The *waiting time* $\tau^{wait}(e_{arr} \to e_{dep})$ of a transfer is $\tau(e_{dep}) - \tau(e_{arr}) - \tau^{walk}(e_{arr} \to e_{dep})$. To model delays that can propagate between different trips, *dependency arcs* $(t_1, t_2) \in \mathcal{D}$ are introduced between two consecutive trips of a vehicle. Like activities, they have a minimum duration. If a trip arrives late at its last stop, the next trip of the vehicle is delayed accordingly.

The agents are generated using an OD-matrix. The OD-matrix specifies how many passengers per hour want to travel from a specific start stop *origin* to a specific destination stop *dest*. Each passenger is assigned a fixed start time τ_{start}. A more realistic modeling, in which the start time is chosen by the agents themselves depending on the network, is conceivable, but is not dealt with in this paper. During a simulated day, a *journey* is created for each passenger, which is an alternating sequence of trip segments and valid transfers. In addition to transfers between two trips, a journey can also have an initial walk at the start or a final walk to the destination. We therefore extend our definition of transfers to include the special cases $\mathit{origin} \to e_{dep}$ and $e_{arr} \to \mathit{dest}$, where *origin* is the start and *dest* is the destination. A final transfer $e_{arr} \to \mathit{dest}$ always has a waiting time of 0. A journey J consisting of n trip segments therefore has the form $J = \{\mathit{origin} \to e^{i_1}_{dep}(t_1), e^{i_1}_{dep}(t_1) \to e^{j_1}_{arr}(t_1), e^{j_1}_{arr}(t_1) \to e^{i_2}_{dep}(t_2), ..., e^{i_n}_{dep}(t_n) \to e^{j_n}_{arr}(t_n), e^{j_n}_{arr}(t_n) \to \mathit{dest}\}$.

3 Agent-Based Dynamic Traffic Assignment Model

In this section we introduce our dynamic traffic assignment model step-by-step. We first sketch and discuss the model of Briem et al. [1], as it serves in many respects as the basis for the simulation presented in this work. Then we present a high-level description of our simulation model. Afterwards, we provide details about the modeling of congestion effects, passenger preferences, route choice, learning, and real-time reactions.

Traffic assignment using Connection Scan Algorithm. In [1], passenger preferences are modeled using perceived arrival times. These are used to make decisions based on factors such as arrival time, number of transfers, walking time, waiting time, and delay robustness. The algorithm simulates different decisions for each passenger (boarding a vehicle, alighting a vehicle, walking to another stop) and assigns probabilities to these options based on the perceived arrival times of each option. The boarding and alighting decisions are binary (board or stay at a stop, alight or stay on a trip). The perceived travel times for all options are calculated with a single run of the Connection Scan Algorithm [7]. In a second scan over all elementary connections, a random decision whether to board the vehicle is made for each passenger waiting at the corresponding departure stop. Then, for all passengers in the current vehicle, a random decision is made whether to alight at the arrival stop. Finally, for each alighting passenger, a random decision is made as to which stop they will walk to (or remain at the current stop). This approach is very efficient, but is based on some unrealistic assumptions. First, the model assumes unlimited vehicle capacities. This leads to traffic allocations in which individual vehicles have unrealistically high load factors. Second, passengers do not react in any way to high occupancy rates and are treated as uniform decision makers; personal preferences or experiences are not implemented. Third, movements of vehicles are not simulated and transfer uncertainty is modeled by adding a random variable to the arrival times. Consequently, it is not possible to model specific vehicle delays and how passengers react to them. As capacities and occupancy rates are ignored and delays are modeled as random variables, the network performance is completely independent from passenger decisions.

3.1 High-level Description of Model

In our model, passengers are modeled as agents with their own preferences and experiences. The individual decisions of the passengers influence the network dynamics and the passenger decisions are in turn dependent on the network dynamics. The network flow is therefore the result of the passengers' interaction with the network. Each individual trip is modeled as a separate entity whose performance depends on the decisions of the agents. The modeling of vehicles as entities allows explicit delays and delay dependencies between vehicles. Section 3.2 describes three different network congestion effects that are implemented in this model: crowded vehicles (including seat allocations), denied boardings and dwell time delays.

The model replicates the impact of network performance on passenger decisions. We develop a flexible choice model that allows passengers to make adaptive decisions in response to these dynamic network conditions. As in [1], we evaluate decisions by calculating a perceived travel time (Section 3.3) for each option, but we incorporate the three modeled congestion effects. In Section 3.4, we explain how probabilities are assigned to different options based on the perceived travel times. Instead of binary choices, passengers choose between boarding trips of different lines and alighting at different downstream stops. The advantage of this is that passengers can react adaptively to the characteristics of different journeys, rather than just considering the current and optimal options. One drawback is that the journey characteristics can change in the time between the decision and the execution of the chosen event. In Section 3.6, we describe how passengers can change their decisions based on real-time information. Individual passengers adapt their behavior to their experiences with the network. Passengers' decisions are influenced by the experience they have gained by repeatedly traveling through the network on consecutive days (Section 3.5). Modeling a learning process is of great importance as the network conditions vary from day to day. The learning process models how passengers react to the fluctuating network congestions on different days and allows them to avoid highly congested trips.

The model is developed as an event-oriented simulation with discrete time steps. We keep the algorithmic framework and the overall structure of a simulated day from the model in [1]. Changes to the algorithm are described in Section 3.3. As event times are no longer static (in particular, due to dwell time delays), the connections cannot be pre-sorted and we need a priority queue Q. This contains all events of the period to be simulated. The events are sorted in ascending order according to the current time of the events. If two events have the same time, arrivals are processed before departures. On each pass through the main loop, the current event is extracted from Q. All passengers who have not yet started their day are loaded into the network. They select their first boarding. The type of event is then distinguished. In the case of a departure event, the passengers waiting at the current stop are processed in random order. In the case of an arrival event, the passengers on the current vehicle are processed. When a departure event is processed, a dwell time delay may be generated and propagated. In this case, Q must be updated. At the end of each day, the expectations and perceived travel times are updated.

3.2 Congestion Effects

Crowded vehicles. Seats are allocated as follows: We assume that passengers alight from the vehicle before those waiting at the stop board. Seats may therefore become available. First, passengers are drawn at random from those currently standing until either all seats are occupied or all passengers are seated. Second, the waiting passengers then board the vehicle in random order. We therefore assume that the passengers mix while waiting. Entering passengers are assumed to take a seat if one is available. The main causes of dissatisfaction in overcrowded vehicles are standing and physical proximity to other passengers. Let $q^{onboard}(e_1, e_2) \leq cap(t)$ be the number of passengers of an activity (e_1, e_2) of the trip t. The discomfort depends on the current passenger *load* $\lambda(e_1, e_2) := q^{onboard}(e_1, e_2)/cap_{sit}(t)$ and on whether a seat has been found. The load is defined relative to the number of seats. These two properties give the *crowding factor* $\beta_k^{crowding}(\lambda(e_1, e_2), seated_k(e_{dep}, (e_1, e_2)))$, where k is the current passenger and e_{dep} is the departure event at which the passenger boarded the current trip. The Boolean function $seated_k$, which indicates whether the passenger k has a seat, thus depends not only on the current arc, but also on the time of boarding. We assume that a passenger will not give up a seat once it has been found. The crowding factor is modeled as a step function (see Table 1). The load of a vehicle is not known to the passenger in advance and is therefore based on the passenger's personal experience on previous days, or a default value if no experience is available. Learning is described in Section 3.5. The model also allows for real-time load information, but this is beyond the scope of this paper.

Denied boardings. As with the seat allocation, the standing room allocation depends on the random order of boarding passengers. It is possible that the number of passengers wishing to board a vehicle is greater than the remaining capacity. In this case, some of the passengers must therefore remain at the stop. We refer to this as *denied boarding* at a departure event e_{dep}. The passengers must respect the minimum transfer time at the current stop and may walk to another stop in response to the denied boarding. They are therefore treated as passengers who alight at $stop(e_{dep})$ at time $\tau(e_{dep})$. Such an unplanned complication is associated with additional stress for the passenger. Therefore, the subsequent waiting or walking time to the next boarding event is penalized by a multiplier β_k^{fail}.

Dwell time delays. We model the dwell time as a monotonically increasing function based on the number of boarding and alighting passengers. Let q^{alight} be the current number of alighting passengers and q^{board} the current number of boarding passengers. The required dwell time is given by $(q^{alight} + q^{board})/doorCapacity(t)$, where $doorCapacity(t)$ is the number of passengers that can board or alight per second. This value differs significantly between different vehicle types, for example buses generally have a smaller $doorCapacity$ than trains. It should be noted that this is a greatly simplified model; for example, the time to open and close the doors is ignored. Since boarding and alighting is the dominant component [22], this is sufficient to capture the systematic evolution of delays caused by the network flow. For more accurate modeling, more information is needed on the vehicles used. If the required dwell time is greater than the scheduled one of a dwelling arc, the corresponding departure is delayed by the difference. Occurring delays are propagated downstream along the corresponding trip (as in [23]). Additionally, delays of trains are propagated across shared rails.

3.3 Perceived Travel Time

Perceived travel time is a key characteristic that influences passenger satisfaction with public transport. Unlike actual travel time, it takes into account that waiting times, walking distances, transfers and in-vehicle crowding are perceived differently by passengers. Each passenger in the model has its own preferences and experiences. The perceived travel time is dependent on the network dynamics of the current day and the passenger's experience gained on previous days. Each passenger has different sources of information about expected times and vehicle loads. These sources can be, in descending order of priority, real-time information, experience, or default values (scheduled times or an input parameter for load). Unless otherwise specified, the source with the highest priority is used.

During the simulation, each passenger is assigned a journey iteratively through partial decisions. Two types of decisions are made: a passenger waiting at a stop, has to decide which trip to board next (or whether to walk directly to the destination if there is a footpath). The passenger chooses between departure events that can be reached from the current stop, including departure events reachable by a footpath. Second, a passenger traveling in a vehicle has to decide at which stop to alight. To implement these decisions, we calculate for each passenger k an expected *perceived travel time* $f_k(e)$ for each boarding and alighting event e, that corresponds to the optimal journey from the current event to the destination. The initial perceived travel times are calculated before the start of the simulation and are based on scheduled times. The perceived travel times are updated at the end of each day after incorporating passenger experience, and are recalculated during a day when real-time information is available. At the time of the decision, the passenger does not know the actual perceived travel times, as the activities are in the future. These values are therefore only estimates for the current day. We account for crowding by weighting the in-vehicle-travel time by a crowding factor $\beta_k^{crowding}$ depending on the vehicle load. Similarly, waiting and walking times are weighted by passenger-specific factors β_k^{walk} and β_k^{wait}, respectively. We also use an additive penalty $\beta_k^{transfer}$ for each transfer and $\hat{\beta}_k^{fail}(tr)$ for a possible failed transfer tr. The penalty term for a failed transfer corresponds to the weighted additional waiting time caused by the failed transfer, multiplied by an estimated probability $p_k^{fail}(tr)$. The expected perceived travel time $f_k(e)$ of an event e is defined recursively as the minimum over all possibilities to continue from this event to the destination. In the following, we derive these calculations step by step. First, we define an expected perceived travel time ptt_k for each transfer and for each trip segment. The perceived travel time of a journey is the sum of the perceived travel times of all trip segments and transfers of the journey (including the waiting time at the origin stop). The perceived travel time of a trip segment is the sum of the perceived travel times of all driving and dwelling arcs of the trip segment. The perceived travel time of an activity (e_1, e_2) is obtained by multiplying the crowding factor with the duration of the activity, i.e.

$$ptt_k(e_{dep}, (e_1, e_2)) := \beta_k^{crowding}(\lambda_k(e_1, e_2), seated_k(e_{dep}, (e_1, e_2))) \cdot \tau_k^{ivt}(e_1, e_2),$$

where e_{dep} is the departure event at which the passenger k boarded the current trip. Since a passenger does not know in advance when he will find a seat, he assumes that he will find a seat at the first arc with an expected load of less than 1. The Boolean function $seated_k(e_{dep}^i(t), (e^j(t), e^{j'}(t)))$ is $true$ if an arc $(e^m(t), e^{m'}(t))$ exists with $\lambda_k(e^m(t), e^{m'}(t)) < 1$ and $i \leq m \leq j$. This is a pessimistic estimate: even if the expected loads match the actually experienced values, the passenger may find a seat earlier.

The perceived travel time of a trip segment is

$$ptt_k(e_{dep}^i(t) \to e_{arr}^j(t)) := \sum_{(e,e') \in activities(e_{dep}^i(t) \to e_{arr}^j(t))} ptt_k(e_{dep}^i(t), (e, e')).$$

The real travel time of a transfer $tr = e_{arr} \to e_{dep}$ consists of the waiting time for the next trip and, if a footpath is required for the transfer, the length of the footpath. These times are multiplied by passenger-specific coefficients β_k^{walk} and β_k^{wait}. In addition, there are penalty terms $\beta_k^{transfer}$ for the transfer itself and $\hat{\beta}_k^{fail}(tr)$ for a possible failed transfer. The penalty term for a failed transfer corresponds to the weighted additional waiting time caused by the failed transfer. Since the additional waiting time after the failed transfer is not known at the time of the decision, it must be estimated. To allow an efficient calculation, this estimate is based only on the timetable. We define the expected additional weighted waiting time after the failed boarding $\hat{\beta}_k^{fail}(tr)$ as $headway(e_{dep}) \cdot \beta_k^{fail}$. The value $\hat{\beta}_k^{fail}(tr)$ is then multiplied by a probability $p_k^{fail}(tr)$ estimated by passenger k that boarding at e_{dep} is not possible due to limited capacity of $trip(e_{dep})$ or delays of e_{arr}. This probability is assumed to be 0 at the beginning of the simulation. It depends on two components, p_k^{denied} and p_k^{delay}. The overall probability $p_k^{fail}(tr)$ is calculated as $p_k^{denied}(e_{dep}) + p_k^{delay}(tr) - (p_k^{denied}(e_{dep}) \cdot p_k^{delay}(tr))$. Both components are based on the passenger's experience on previous days. The probability p_k^{delay} is calculated using a weighted empirical distribution function of the arrival times $\tau(e_{arr})$. The perceived travel time of a transfer $tr = e_{arr} \to e_{dep}$ is

$$ptt_k(tr) := \hat{\beta}_k^{fail}(tr) \cdot p_k^{fail}(tr) + \beta_k^{wait} \cdot \tau_k^{wait}(tr) + \beta_k^{walk} \cdot \tau^{walk}(tr) + \beta_k^{transfer}.$$

We additionally define $walk_k^{dest}(s) = \beta_k^{walk} \cdot \ell(s, dest)$ as the weighted walking time from stop s to destination $dest$ of passenger k. Using these definitions, we can now recursively define the expected perceived travel times $f_k(e)$. The minimum perceived travel time from an arrival event e_{arr} to the destination stop is given by the minimum over all outgoing valid transfers and the weighted walking time to the destination stop. Transfers that are only possible due to a learned delay of the departure event are ignored here. We first define the minimum over all transfers:

$$f_k^{trans}(e_{arr}) := \min_{e_{arr} \to e'_{dep}} ptt_k(e_{arr} \to e'_{dep}) + f_k(e'_{dep}).$$

The value $f_k(e_{arr})$ is the minimum of this value and the weighted walking time, i.e.

$$f_k(e_{arr}) = \min\{f_k^{trans}(e_{arr}), walk_k^{dest}(stop(e_{arr}))\}.$$

In particular, $f_k(e_{arr})$ is 0 if $stop(e_{arr})$ is the destination of k. The minimum perceived travel time of a departure event e_{dep} is the minimum over all possible trip segments. The respective trip segment is defined by the arrival event at which the passenger alights. For a departure event $e^i_{dep}(t)$ this results in

$$f_k(e^i_{dep}(t)) = \min_{j>i} ptt_k(e^i_{dep}(t) \to e^j_{arr}(t)) + f_k(e^j_{arr}(t)).$$

We have therefore defined a minimum perceived travel time $f_k(e)$ to the destination for each passenger k and for each possible boarding and alighting event.

Calculation of initial perceived travel times. Algorithm 1 describes the calculation of the initial perceived travel times $f_k(e)$. These initial values are independent of the network dynamics. The times therefore correspond to the regular times according to the timetable, a standard load λ_{std} is assumed for each activity and the probability of a failed boarding is assumed to be 0. The algorithm is based on the profile connection scan algorithm [7], with the difference that we calculate perceived travel times instead of earliest arrival times. In addition, dwelling activities must also be taken into account. As in [7], we first perform a simple earliest arrival time query with CSA to determine the driving arcs C that can be reached from the origin. We limit the time horizon to $\tau_{arr}(k) + \Delta_\tau$, where $\tau_{arr}(k)$ is the earliest (real) arrival time of k at its destination *dest*. We therefore discard journeys that arrive more than Δ_τ later at the destination than the fastest journey. During the execution of the algorithm, a set of Pareto-optimal journeys $B[s]$ from s to the destination is calculated for each stop $s \in \mathcal{S}$. The criteria are the departure time and the minimum perceived travel time to the destination. For each departure event e_{dep}, a label $L = (\tau_{dep}, ptt, trip)$ is created consisting of the departure time τ_{dep}, the minimum perceived travel time to the destination *ptt* and the first trip of the journey *trip*. For Pareto dominance, the difference between the departure times of the labels must be added to the perceived travel time of the later label.

We iterate over the driving arcs C in descending order of departure times. In each loop iteration, the invariant applies that $ptt_{curr}[t]$ is the minimum perceived travel time from the earliest scanned departure event of the trip t to the destination. At the beginning of the iteration for the travel arc $(e^i_{dep}(t), e^{i+1}_{arr}(t))$, $ptt_{curr}[t]$ therefore corresponds to the minimum perceived travel time from $e^{i+1}_{dep}(t)$ to the destination. At the arrival event $e^{i+1}_{arr}(t)$, the passenger has three options: he can change to another trip, walk to the destination or stay in the vehicle. We calculate the minimum of these three options. First, we calculate the perceived travel time for a transfer. The optimal transfer is the transfer to the Pareto-optimal partial journey with the smallest departure time. Here we discard the case that a passenger alights to get back on the current trip immediately. Let L_f be the label of this journey. The perceived travel time for a transfer $transfer$ is therefore the sum of the perceived travel time of L_f and the cost of the transfer (waiting time and penalty for transfer). The perceived

Algorithm 1 Calculation of the initial perceived travel times f_k to the destination of passenger k.

```
Input: list $C$ of relevant driving arcs sorted by regular departure times
Output: perceived travel times $f_k(e)$
foreach $t \in \mathcal{T}$ do
    $ptt_{curr}[t] \leftarrow \infty$
foreach driving arc $(e^i_{dep}(t), e^{i+1}_{arr}(t)) \in C$, descending by $\tau(e^i_{dep}(t))$ do
    let $L_{transfer}$ be the label $L \in B[stop(e^{i+1}_{arr}(t))]$ with minimum departure time
      $\tau_{dep}(L)$, for which $\tau_{dep}(L) \geq \tau(e^{i+1}_{arr}(t))$ and $trip(L) \neq t$ apply
    if $L_{transfer} \neq \perp$ then
        $transfer \leftarrow ptt(L_{transfer}) + \beta^{transfer}_k + \beta^{wait}_k(\tau_{dep}(L_{transfer}) - \tau(e^{i+1}_{arr}(t)))$
    else
        $transfer \leftarrow \infty$
    $alight \leftarrow \min\{transfer, walk^{dest}_k(stop(e^{i+1}_{arr}(t)))\}$
    $remain \leftarrow ptt_{curr}[t] + \beta^{crowding}_k(\lambda_{std}, \lambda_{std} < 1) \cdot \tau^{ivt}(e^{i+1}_{arr}(t), e^{i+1}_{dep}(t))$
    $minptt \leftarrow \min\{alight, remain\}$
    if $minptt = \infty$ then continue
    $ptt_{curr}[t] \leftarrow minptt + \beta^{crowding}_k(\lambda_{std}, \lambda_{std} < 1) \cdot \tau^{ivt}(e^i_{dep}(t), e^{i+1}_{arr}(t))$
    $f_k(e^{i+1}_{arr}(t)) \leftarrow alight$
    $f_k(e^i_{dep}(t)) \leftarrow ptt_{curr}[t]$
    $L_s \leftarrow (\tau(e^i_{dep}(t)) - mct(stop(e^i_{dep}(t))), ptt_{curr}[t] + \beta^{wait}_k \cdot mct(stop(e^i_{dep}(t))), t)$
    if $L_s$ is not dominated: insert $L_s$ into $B[stop(e^i_{dep}(t))]$ and remove dominated
      labels
    foreach footpath $(s', stop(e^i_{dep}(t)) \in \mathcal{F}$ do
        $L_f \leftarrow (\tau(e^i_{dep}(t)) - \ell(s', stop(e^i_{dep}(t)), ptt_{curr}[t] + \beta^{walk}_k \cdot \ell(s', stop(e^i_{dep}(t)), t)$
        if $L_f$ is not dominated: insert $L_f$ into $B[s']$ and remove dominated labels
```

travel time for the passenger to walk to the destination is given by $walk^{dest}_k(stop(e^{i+1}_{arr}(t)))$. We summarize these two options under *alight*. The perceived travel time for staying in the vehicle is equal to the sum of the minimum perceived travel time from $e^{i+1}_{dep}(t)$ to the destination (i.e. $ptt_{curr}[t]$) and the cost of the dwelling arc between $e^{i+1}_{arr}(t)$ and $e^{i+1}_{dep}(t)$. The sum of these two costs is called *remain*. The minimum of all three options is *minptt*.

Afterwards, $ptt_{curr}[t]$ is updated. The minimum perceived travel time for the departure event $e^i_{dep}(t)$ corresponds to the sum of the costs of the current driving arc and the costs of the minimum option at the arrival event (*minptt*). We store the calculated minimum perceived travel times in f_k. We still need to update the Pareto sets. We create a label for the current stop and for all footpaths. We incorporate the minimum transfer time or footpath length directly into the labels. The departure time of the label therefore corresponds to the departure time of $e^i_{dep}(t)$ minus the minimum transfer time or walking distance. For the perceived travel time *ptt* of the labels, the corresponding costs for waiting or walking must be added to the current perceived travel time $ptt_{curr}[t]$. We first test whether a label is dominated by another label of the corresponding Pareto set. If this is not the case, it is inserted. The labels dominated by the inserted label are then removed.

Updating the perceived travel times. Updating the perceived travel times f_k works in a similar way to the initial calculation in Algorithm 1. This subsection only describes the differences. Only a small subset of all events is affected by the update. It would therefore be suboptimal to scan the entire list C. Instead, we use a priority queue Q, which contains the arcs in descending order according to regular departure times. At the beginning, Q contains the arcs for which at least one property has changed. At the end of each iteration, all driving arcs through which the current driving arc can be reached are inserted into Q. These are the previous driving arc of the current trip t and all driving arcs $(e^j_{dep}(t'), e^{j+1}_{arr}(t'))$ for which the transfer $e^{j+1}_{arr}(t') \to e^i_{dep}(t)$ is valid. Another difference is that during the update, each stop is usually visited much less frequently, as only a small subset of the travel arcs are scanned. We therefore do not calculate the Pareto sets. Instead, we calculate the minimum perceived travel time for a transfer by scanning over all departure events e'_{dep} that are reachable from $e^{i+1}_{arr}(t)$ via a valid transfer. Transfers that are only valid because of the learned delay of the boarding event are ignored. We therefore use the regular time for e'_{dep}.

In the initial calculation, a default value λ_{std} was assumed for the load of each arc. This means that a passenger assumes that they are either always seated or always standing. This is no longer the case with the update. As a reminder: A passenger assumes that they must stand until they reach an arc $(e^i_{dep}(t), e^{i+1}_{arr}(t))$ for which they expect a load factor of less than 1, i.e. $\tilde{\lambda}_k(e^i_{dep}(t), e^{i+1}_{arr}(t)) < 1$. In addition to $ptt_{curr}[t]$, we store another value $ptt^{sitting}_{curr}[t]$, which is the minimum perceived travel time to the destination, assuming that the passenger is seated for the rest of the journey. If we scan an arc $(e^i_{dep}(t), e^{i+1}_{arr}(t))$ with $\tilde{\lambda}_k(e^i_{dep}(t), e^{i+1}_{arr}(t)) < 1$, we set $ptt_{curr}[t] \leftarrow ptt^{sitting}_{curr}[t]$. The initial calculation of the minimum perceived travel times and their update at the end of the day or in real-time reactions is independent of the other passengers. The parallelization of these steps is therefore trivial.

3.4 Choice Model

We use a mixed $(\epsilon - greedy, SoftMax)$ decision model. With a probability of $1 - \epsilon$, the optimal decision is made directly. In the other case, the SoftMax selection is used. This mixed model results in the agents predominantly making the optimal decision, while occasionally opting for a random action according to the SoftMax principle. The SoftMax function is used to assign probabilities to the individual decisions based on the perceived travel times f_k.

In general, the SoftMax selection for any actions a with costs $f(a)$ has the following form:

$$p(a) := \frac{e^{(f(a^{opt}) - f(a))/\gamma(d)}}{\sum\limits_{a'} e^{(f(a^{opt}) - f(a'))/\gamma(d)}},$$

where a^{opt} is the optimal action, $\gamma(d)$ is the temperature and d is the current day. The costs of an action are therefore considered relative to the optimal costs. If a high temperature is chosen, the probability of making suboptimal decisions is higher. In the limit value for $\gamma \to 0$, the optimal decision is always made. The temperature therefore influences the average perceived travel times of passengers.

Boarding and walking decisions. When a passenger k is waiting at a stop, he decides on a trip, specifically a departure event, which he wants to board next. For simplicity, assume that the passenger just alighted at an arrival event e_{arr}. The passenger decides on the basis of the perceived travel time f_k. We restrict the departure events in question to the earliest available trips on each line. Let the set of relevant departure events be $reldep(e_{arr}) := \{e^i_{dep}(t) | e_{arr} \to e^i_{dep}(t)$ is valid and there is no valid transfer $e_{arr} \to e^i_{dep}(t')$ with $line(t) = line(t')$ and $\tau(e^i_{dep}(t')) < \tau(e^i_{dep}(t))\}$. Here we only consider transfers if they are valid for the regular time $\tau_{reg}(e_{dep})$ of the departure event. Transfers that become valid due to delays are handled as part of the real-time reactions in Section 3.6. The expected perceived travel time for a selected boarding event e_{dep} consists of the perceived travel time of the transfer $e_{arr} \to e_{dep}$ and $f_k(e_{dep})$. As we only consider valid transfers and the arrival time $\tau(e_{arr})$ is fixed at the moment of the decision, the probability $p_k^{delay}(e_{arr} \to e_{dep})$ of the transfer being invalid because of a delay is 0. Let

$$f_k^*(e_{arr}) := \min_{e_{arr} \to e_{dep}, e_{dep} \in reldep(e_{arr})} ptt_k(e_{arr} \to e_{dep}) + f_k(e_{dep})$$

be the perceived travel time for the optimal transfer among the relevant ones. The passenger first decides whether to walk to the destination or wait for a ride. The perceived travel time of the optimal transfer $f_k^*(e_{arr})$ is compared with the weighted walking time $walk_k^{dest}(s)$. Let $f_k^{opt} := \min(walk_k^{dest}(s), f_k^*(e_{arr}))$ be the optimal decision. The probability that the passenger decides to walk to the destination is

$$p(walk) := \frac{e^{(f_k^{opt} - walk_k^{dest}(s))/\gamma(d)}}{e^{(f_k^{opt} - walk_k^{dest}(s))/\gamma(d)} + e^{(f_k^{opt} - f_k^*(e_{arr}))/\gamma(d)}},$$

If the passenger does not walk to the destination, he chooses a departure event from $reldep(e_{arr})$. We define the relative perceived travel time for a transfer $e_{arr} \to e_{dep}$ with $e_{dep} \in reldep(e_{arr})$ as

$$f_k^{rel}(e_{arr} \to e_{dep}) := f_k^*(e_{arr}) - (ptt_k(e_{arr} \to e_{dep}) + f_k(e_{dep})).$$

This value is therefore 0 for the optimal transfer and negative otherwise. The probability that the passenger decides for the transfer $e_{arr} \to e_{dep}$ is

$$p(e_{arr} \to e_{dep}) := \frac{e^{f_k^{rel}(e_{arr} \to e_{dep})/\gamma(d)}}{\sum\limits_{e'_{dep} \in reldep(e_{arr})} e^{f_k^{rel}(e_{arr} \to e'_{dep})/\gamma(d)}}.$$

If a walk is required for the selected transfer, the passenger changes the stop after making the decision. So far, we have assumed that the passenger has just alighted at an arrival event e_{arr}. However, it is also possible that a departure decision is made that does not immediately follow an alighting event. This is not a problem because any tuple (s, τ) consisting of a stop and a time can define a set of relevant boardings. We described the special case $(stop(e_{arr}), \tau(e_{arr}))$.

Alighting decisions. If a passenger has boarded at a departure event $e_{dep}^i(t)$ of the trip t, he decides at which downstream stop of t he will alight. He therefore decides on an arrival event $e_{arr}^j(t)$ with $j > i$. Together, $e_{dep}^i(t)$ and $e_{arr}^j(t)$ result in a trip segment $e_{dep}^i(t) \to e_{arr}^j(t)$ of the journey. The expected perceived travel time for a selected exit results from the sum of the perceived travel time of this trip segment and $f_k(e_{arr}^j(t))$. We again define the perceived travel time relative to the optimal decision. The perceived travel time for the optimal alighting decision is

$$f_k^*(e_{dep}^i(t)) := \min_{m>i} ptt_k(e_{dep}^i(t) \to e_{arr}^m(t)) + f_k(e_{arr}^m(t)).$$

The relative perceived travel time for an exit at the jth stop is

$$f_k^{rel}(e_{dep}^i(t) \to e_{arr}^j(t)) := f_k^*(e_{dep}^i(t)) - (ptt_k(e_{dep}^i(t) \to e_{arr}^j(t)) + f_k(e_{arr}^j(t))).$$

This value is 0 for the optimal decision and negative otherwise. The probability that the passenger chooses the jth stop is

$$p(e_{dep}^i(t) \to e_{arr}^j(t)) := \frac{e^{f_k^{rel}(e_{dep}^i(t) \to e_{arr}^j(t))/\gamma(d)}}{\sum\limits_{m>i} e^{f_k^{rel}(e_{dep}^i(t) \to e_{arr}^m(t))/\gamma(d)}}.$$

3.5 Learning

After each simulated day, the expected perceived travel times f_k are updated according to the experiences on the current day. Thus, an explicit learning process is modeled through which the passengers learn properties of the network. We mark the learned properties with a tilde. The properties learned are the loads of driving arcs $\tilde{\lambda}_k(e_{dep}, e_{arr})$, the probabilities for denied boardings $\tilde{p}_k^{denied}(e_{dep})$, the probabilities for failed transfers because of delays $\tilde{p}_k^{delay}(e_{arr} \to e'_{dep})$ and the times $\tilde{\tau}_k(e)$ of the events. The learned values are updated in two steps: First, the expected properties are updated for the activities and events experienced by the passenger on the current day. Then the perceived travel times f_k are recalculated. The recency parameter κ indicates how highly new experiences are weighted. For $\kappa = 1$, the accumulated experiences correspond to the average of all experiences. For $\kappa < 1$ newer experiences are weighted higher and for $\kappa > 1$ lower. The accumulated experiences are defined as in [4] as a function of the experiences on the current day and the accumulated experiences before the current day. Since not every event is updated every day, we need to memorize the number of updates for each event and property. Let $|\lambda_k(e_{dep}, e_{arr})|, |p_k^{denied}(e_{dep})|$ and $|\tau_k(e)|$ be the number of updates. The respective number is incremented before the update.

At the end of each day, a passenger k has a journey $J_k = \{o \to e^{i_1}_{dep}(t_1), e^{i_1}_{dep}(t_1) \to e^{j_1}_{arr}(t_1), e^{j_1}_{arr}(t_1) \to e^{i_2}_{dep}(t_2), ..., e^{i_n}_{dep}(t_n) \to e^{j_n}_{arr}(t_n), e^{j_n}_{arr}(t_n) \to dest\}$ and a set of denied boardings D_k. Let $driving\_arcs(J_k)$ be the set of all driving arcs of J_k, $events(J_k)$ the set of all events of J_k and $events_{dep}(J_k)$ the set of all departure events of J_k. In addition to the times of the events and the utilization of the activities, the proportion of passengers who were unable to board due to limited capacity out of those who attempted to on the current day is also stored for each departure event. Let $p_d^{denied}(e_{dep})$ be this quantity.

Let $\lambda_d(e_{dep}, e_{arr}), p_d^{denied}(e_{dep})$ and $\tau_d(e)$ be the respective properties of the network on day d. The first step is to integrate the properties of the current day into the learned values $\tilde{\lambda}_k(e_{dep}, e_{arr}), \tilde{p}_k^{denied}(e_{dep})$ and $\tilde{\tau}_k(e)$.

We update the load $\tilde{\lambda}_k(e_{dep}, e_{arr})$ for the driving arcs $(e_{dep}, e_{arr}) \in driving\_arcs(J_k)$:

$$\tilde{\lambda}_k(e_{dep}, e_{arr}) \leftarrow \tilde{\lambda}_k(e_{dep}, e_{arr}) \cdot (1 - |\lambda_k(e_{dep}, e_{arr})|^{-\kappa}) + \lambda_d(e_{dep}, e_{arr}) \cdot |\lambda_k(e_{dep}, e_{arr})|^{-\kappa},$$

for the departure events $e_{dep} \in events(J_k) \cup F_k$ the probabilities $\tilde{p}_k^{denied}(e_{dep})$:

$$\tilde{p}_k^{denied}(e_{dep}) \leftarrow \tilde{p}_k^{denied}(e_{dep}) \cdot (1 - |p_k^{denied}(e_{dep})|^{-\kappa}) + p_d^{denied}(e_{dep}) \cdot |p_k^{denied}(e_{dep})|^{-\kappa},$$

and for the events $e \in events(J_k)$ the times $\tilde{\tau}_k(e)$:

$$\tilde{\tau}_k(e) \leftarrow \tilde{\tau}_k(e) \cdot (1 - |\tau_k(e)|^{-\kappa}) + \tau_d(e) \cdot |\tau_k(e)|^{-\kappa}.$$

We set the learned load of a dwelling arc $(e^i_{arr}(t), e^i_{dep}(t))$ to the learned load of the following driving arc, i.e. $\tilde{\lambda}_k(e^i_{arr}(t), e^i_{dep}(t)) \leftarrow \tilde{\lambda}_k(e^i_{dep}(t), e^{i+1}_{arr}(t))$. In order to learn the probability $\tilde{p}_k^{delay}(e_{arr} \to e_{dep})$ that a transfer fails because e_{arr} is delayed, the experienced times for each arrival event in $events(J_k)$ are memorized. Let $\mathcal{T}^d(e_{arr})$ be the sampled arrival times of e_{arr} after day d. We calculate an inverse weighted empirical distribution function based on this sample. The probability $\tilde{p}_k^{delay}(e_{arr} \to e_{dep})$ is equal to the weighted share of sampled arrival times that are greater than $\tau_{reg}(e_{dep}) - \tau_{min}(e_{arr} \to e_{dep})$, i.e.

$$\tilde{p}_k^{delay}(e_{arr} \to e_{dep}) = \sum_{i=1}^{d} w_i^d \cdot \mathbb{1}_{\mathcal{T}_i > \tau_{reg}(e_{dep}) - \tau_{min}(e_{arr} \to e_{dep})},$$

where $\tau_{min}(e_{arr} \rightarrow e_{dep})$ is the minimum required time for the transfer (either minimum change time or footpath length) and $\mathbb{1}_{\mathcal{T}_i > \tau_{reg}(e_{dep}) - \tau_{min}(e_{arr} \rightarrow e_{dep})}$ is the indicator for $\mathcal{T}_i > \tau_{reg}(e_{dep}) - \tau_{min}(e_{arr} \rightarrow e_{dep})$ and

$$w_i^d = i^{-\kappa} \cdot \prod_{j=i+1}^{d} (1 - j^{-\kappa}).$$

These normalized weights are derived from the update formulas above. A problem arises when updating the times: as only parts of a trip are updated, it may happen that $\tilde{\tau}_k(e_1) > \tilde{\tau}_k(e_2)$ applies to an activity (e_1, e_2). The property that the times of a trip are non-decreasing is therefore violated. To restore this property, the delays at the first or last event are extrapolated to the start or end of each trip.

After the values $\tilde{\lambda}_k, \tilde{p}_k^{denied}, \tilde{p}_k^{delay}$ and $\tilde{\tau}_k$ have been updated, the expected perceived travel times f_k must be updated. However, a complete recalculation is not necessary as only a small subset of all events and activities have changed.

3.6 Real-time Reactions

Until now, passengers' decisions have been based on personal experience or, in the absence of experience, on the timetable and default values. However, current circumstances may differ from these experiences. Therefore, mechanisms are implemented that allow adaptive behavior based on current information. Similar to Milatras [27], these mechanisms allow to change the original decision. In general, a new decision is made if the currently selected action is worse than expected or if an unselected action is better than expected. The affected perceived travel times $f(e)$ are updated with a CSA query, taking into account the current information. The new decision then follows the same principle as the original, but with updated scores for each option. Boarding and alighting decisions are reconsidered for various cases, mainly due to differences between expected and actual event times or vehicle loads.

We distinguish between real-time reactions while a passenger is waiting at a stop and real-time reactions while a passenger is in a vehicle. For real-time reactions at stops, we further distinguish between two cases. In the first case, let the originally selected trip t^* be the currently departing trip (boarding_redo). If $\tau(e^i_{dep}(t^*)) > \tilde{\tau}_k(e^i_{dep}(t^*))$ applies, the trip is more delayed than expected and a change of decision is possible. After updating $f_k(e^i_{dep}(t^*))$, a completely new boarding decision is made as described in Section 3.4. The current trip t^* remains a possible option.

For the second case, let $e^j_{dep}(t) \neq e^i_{dep}(t^*)$. The passenger therefore has the option of switching from the currently selected trip t^* to trip t (boarding_switch). We distinguish between four different reasons why a switch could be advantageous: (1) trip t is departing earlier than expected, (2) there is still free capacity in t and $\tilde{p}_k^{denied}(e^j_{dep}(t)) > 0$ applies, (3) boarding at $e^j_{dep}(t)$ is not possible according to regular times and was therefore not included in the original decision, and (4) the current simulation time τ_{curr} is greater than the expected departure time of the current choice t^*, i.e., the current choice is more delayed than the passenger expected.

A binary decision between these two options is made following the described $(\epsilon - greedy, SoftMax)$ decision model with two restrictions after updating the values $f(e^j_{dep}(t))$ and $f(e^i_{dep}(t^*))$. To avoid a bias towards earlier departing but worse trips, the switch is only executed if the updated perceived travel time for $e^j_{dep}(t)$ is not worse than the updated perceived travel time for $e^i_{dep}(t^*)$. On the other hand, we also want to avoid that the passenger switches to a better trip if the original choice was already suboptimal, as this would defeat the purpose of a probabilistic decision model. The switch is therefore not made if the original choice was suboptimal and the original choice is not worse than expected under the current circumstances.

The real-time reactions in vehicles work in a similar way to the real-time reactions at stops. The current passenger k has a currently selected arrival event $e^{i^*}_{arr}(t)$ at which they want to alight. If the trip t arrives at a stop, the passenger has the option to change his decision. Let $e^{i}_{arr}(t)$ with $i \leq i^*$ be the current arrival event. In the case where $i = i^*$, the passenger has arrived at the stop where he plans to alight. A change of decision is possible if the current trip is more delayed than expected. If the optimal transfer is still possible, no change of decision is necessary, as the journey with the minimum perceived travel time to the destination remains the same. If this transfer is no longer possible, the perceived travel time for an exit at $e^{i}_{arr}(t)$ becomes worse. In this case, a new alighting decision is made, taking into account the current information (alighting_redo). The current arrival remains a possible option.

If $i < i^*$ applies, the passenger has the option of alighting early at the current stop (alighting_switch). There are three reasons why early alighting might be attractive: (1) the current trip t arrived earlier than expected, which could allow a new transfer at the current stop, (2) the current trip t arrived later than expected and the optimal transfer at the originally selected exit at $e^{i^*}_{arr}(t)$ is no longer possible with the projected downstream delay, and (3) the passenger has no seat and $seated_k(e^{h}_{dep}(t), (e^{i-1}_{dep}(t), e^{i}_{arr}(t))$ was *true* at the time of the original decision ($e^{h}_{dep}(t)$ being the boarding event), in which case the passenger expects to have to stand for the rest of the trip. If one of these causes is given, a binary decision is made between the two events $e^{i}_{arr}(t)$ and $e^{i^*}_{arr}(t)$. As with the decision at stops, the switch is only made if the expected perceived travel time for alighting at the current event $e^{i}_{arr}(t)$ is not worse than for the current choice $e^{i^*}_{arr}(t)$. If the original choice $e^{i^*}_{arr}(t)$ was suboptimal and the optimal transfer is still possible after the exit at event $e^{i^*}_{arr}(t)$ according to the current information, the switch is also not made. It is still possible for a passenger to find a seat sooner than they expected when they made their original decision. In this case, later exits can become more attractive because the perceived travel time in the current trip is smaller than expected. If the passenger finds a seat, a new alighting decision is made.

4 Experimental Study

In this section, the model is tested on two different public transportation networks. First, the experimental setup and the choice of parameters are presented. Subsequently, three different experiments are conducted to assess the proposed model.

4.1 Experimental Setup

The simulation was implemented in C++ and compiled with mvc 14.3 on Windows 10 using the O2 compiler option. The experiments were run on an AMD Ryzen 7 5800X, clocked at 4.7 GHz during program execution, with 32 GB DDR4 memory with a latency of CL16 and a clock frequency of 3600 MHz. As the model is probabilistic, the results are averaged over ten runs. We simulate a total of two hours of the timetable per day and evaluate all passengers who start their journey within the first simulated hour. For each run, we simulate 30 days. The 2-hour timeframe is sufficient to demonstrate the learning process as the timetable is hourly periodic. As we only simulate a limited timeframe, it is not guaranteed that passengers can finish their journey, for example in the case of consecutive denied boardings. For this case, we introduce a new penalty $\beta^{unfinished}$, which we set to the Euclidean distance in

Table 1 Crowding factor $\beta_k^{crowding}$.

load	seated	standing
$0 \leq \lambda \leq 0.6$	1.0	
$0.6 < \lambda \leq 1.0$	1.2	
$1.0 < \lambda \leq 2.0$	1.4	2.2

Table 2 Characteristics of the Stuttgart and Göttingen networks. The number of trips and driving arcs refer to the 2-hour simulation frame.

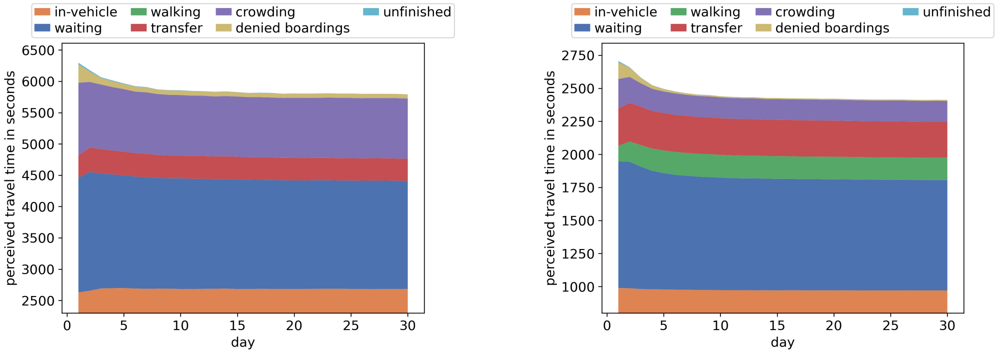
Network	#stops	#lines	#trips	#driving arcs	#footpaths	# passengers per hour
Göttingen	257	22	205	2348	0	1943
Stuttgart	735	403	3175	17381	8732	44836

meters between the current stop and the destination. The parameters for the perceived travel times are identical for each passenger k and are chosen as $\beta_k^{wait} = 1$, $\beta_k^{walk} = 1.5$ and $\beta_k^{transfer} = 300$. We choose $\beta_k^{fail} = 2$ as the multiplier for the additional waiting or walking time after a failed boarding. The choice of the crowding factor $\beta_k^{crowding}$ is based on a British meta-study [28] and is summarized in Table 1.

We use datasets for the public transportation networks of Göttingen (goevb) and Stuttgart [8], provided in LinTim format [23], including the OD matrix. Stuttgart is a mixed network, consisting of 25 train lines and 378 bus lines. Göttingen, in contrast, is a pure bus network consisting of 22 lines. Table 2 shows the main properties of the two data sets. The buses in Stuttgart have a total capacity of 70 and the total capacity of the trains is between 400 and 1000. In Göttingen, buses have a capacity of 50. For simplicity, we assume that the number of seats corresponds to half the total capacity. This is in line with common bus models. We assume that for buses 0.4 passengers can board or alight per second ($doorCapacity$), corresponding to the value recommended by the Transportation Research Board [22]. For trains, let $doorCapacity = cap/200$. The value is chosen depending on the capacity, based on a study for the French city of Nantes [6]. The minimum transfer times mct are specified by LinTim for both data sets. For Stuttgart, the minimum transfer time is $60s$ and for Göttingen $180s$. Footpaths were limited to a maximum of $1800s$. We discard journeys that arrive more than $\Delta_\tau = 3600s$ later than the fastest journey on the current day. The rest of the parameters have been determined by testing. We have chosen a constant temperature of $\gamma = 400$ for Göttingen and $\gamma = 250$ for Stuttgart. For both networks, we chose $\epsilon = 0.2$ for the probability of choosing a random option. Furthermore, we have set the recency parameter to $\kappa = 0.5$ and the standard load to $\lambda_{std} = 0.5$.

4.2 Performance

The simulation was run in parallel on 16 threads. For Stuttgart, an average runtime of 493 seconds was achieved for the whole simulation. Calculating the initial perceived travel times took 70 seconds. On a single day, 89672 passengers were simulated in less than 14.1 seconds on average. For Göttingen, the whole simulation took about 6 seconds. The memory consumption of the simulation is relatively high, as each passenger has to store their personal experiences and expected travel times. For Stuttgart 10.5 GB and for Göttingen 0.2 GB were used.



(a) Göttingen.

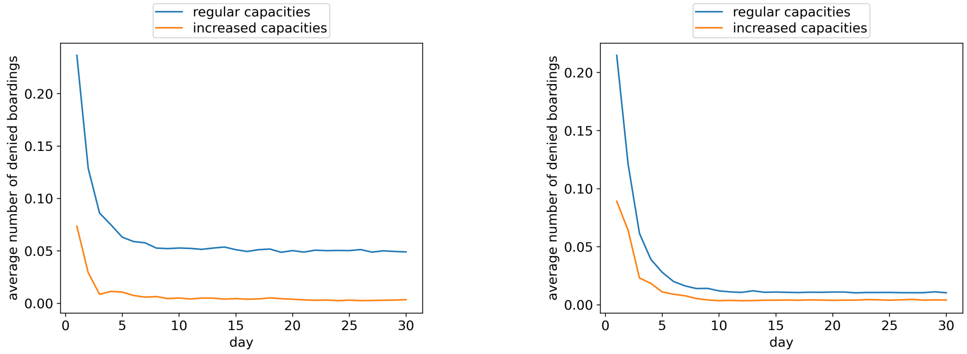
(b) Stuttgart.

Figure 1 Evolution and composition of the perceived travel times.

4.3 Experiments

Experiment 1: Evolution of perceived travel times. In a first experiment, we examine the evolution of the average perceived travel times over 30 days. This evolution is shown for both networks in Figure 1, including the components of the perceived travel times. For simplicity, we assume that all passengers start their learning process on the first day. More complex scenarios are possible in the model. The overall average perceived travel time decreases from 6296s to 5792s for Göttingen and from 2707s to 2412s for Stuttgart. Perceived travel times decrease significantly in the first few days. After 10 days, only minimal gains are achieved. The variance between each of the ten runs was quite small with a maximum deviation from the mean improvement of about 10% for Göttingen and 6% for Stuttgart. The three effects of network congestion are directly reflected in the additional waiting time for denied boardings, the real waiting time and the penalty term for overcrowded vehicles. For both networks, these components of the total perceived travel time decrease over the course of the simulation. The greatest difference is recorded in the additional waiting time penalty for denied boardings. For Göttingen, this value decreases by 227s and for Stuttgart by 118s. The average number of denied boardings per passenger decreases from 0.24 to 0.05 for Göttingen and from 0.21 to 0.01 for Stuttgart. For both networks, the real waiting time decreases significantly during the simulation (108s decrease for Göttingen and 123s decrease for Stuttgart). For Göttingen, it increases on the first few days and only drops below the initial value on the fourth day. The reason for this is that passengers explore nominally worse journeys due to the experienced network congestion effects. Similar effects can be seen in the number of transfers, walking time and the real travel time. If the nominally worse journeys are similarly congested as the journeys selected on previous days, the overall perceived travel time can increase. The crowding penalty decreases by 201s for Göttingen and 64s for Stuttgart. Most of this improvement is due to passengers' desire to avoid standing. The average standing time drops from 495s to 307s for Göttingen and from 69s to 31s for Stuttgart. In this experiment, we have shown how passengers respond to network congestion through the learning process and improve their average perceived travel time by incorporating personal experiences and avoiding congestion. We have found that denied boardings and the resulting additional waiting times have the largest impact on passengers.

Experiment 2: Capacity expansion. In this experiment, we examine the benefits of increasing capacity, targeted to trips operating on full capacity. To determine candidates for a capacity increase, we simulate both networks for one day and determine the trips



(a) Göttingen.

(b) Stuttgart.

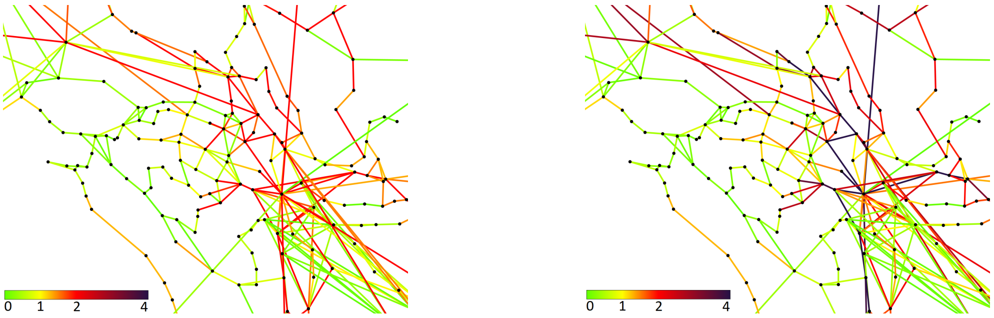
Figure 2 Evolution of denied boardings per passenger with regular and increased capacities.

that are affected by denied boardings. Each trip with at least one denied boarding has its capacity increased by 40%. As a result, the capacities were increased for 41 trips in Göttingen and for 236 trips in Stuttgart. A large benefit was achieved by this capacity expansion for both networks, especially on the first day of the simulation. Compared to the first day of Experiment 1, the total perceived travel time for Göttingen decreased from 6296s to 5775s. For Stuttgart, the difference is smaller (from 2707s to 2585s). As the simulation progresses, this difference becomes smaller as passengers adjust their behavior when capacity is at its limit. At the end of the simulation, the difference compared to Experiment 1 is 321 seconds for Göttingen and 31 seconds for Stuttgart. The improvement is largely due to the reduction in the number of denied boardings and the resulting improvement in real waiting times. The average number of denied boardings is shown in Figure 2. On the final day, the average number of denied boardings is below 0.01 for both networks, which is a substantial improvement for Göttingen. This suggests that for Göttingen the regular capacity is too limited to satisfy passenger demand.

Experiment 3: Unlimited capacities. In Experiment 3, we study the scenario of unlimited vehicle capacities. As a result, some trips are highly overloaded. For Stuttgart, there are buses with over 300 passengers and trains with over 1000 passengers. Similarly, buses with over 130 passengers are found in Göttingen. For Göttingen, 7.9% of all driving edges have loads greater than their regular capacity, compared to 2.6% for Stuttgart. In Figure 3, we compare the vehicle load of normal capacities (left) and unlimited capacities (right). The color coding is relative to the seat capacity. The value 2.0 corresponds to full capacity and 1.0 to full seat capacity. Values above 2.0 indicate overload. We observe that similar segments of the network have high loads in both cases, but considering capacities avoids overloading.

5 Conclusions and Outlook

We presented a fine-grained framework for a dynamic agent-based simulation of traffic assignment in public transit networks. This model is extendible to include real-time delay information or real-time load rates. First experimental studies with our prototype prove to be highly efficient for simulation tests with medium-sized metropolitan regions. As part of future work, further case studies are needed to assess the model validity and scalability to even larger networks as well as to calibrate model parameters.



(a) normal vehicle capacities.

(b) unlimited vehicle capacities.

Figure 3 Small excerpt from the Stuttgart network. Comparison of vehicle loads: normal (left) vs. unlimited vehicle capacities (right). The maximum capacity utilization on the respective segment is shown. The color coding is relative to the seat capacity. The value 2.0 corresponds to full capacity and 1.0 to full seat capacity.

References

1 L. Briem, S. Buck, H. Ebhart, N. Mallig, B. Strasser, P. Vortisch, D. Wagner, and T. Zündorf. Efficient Traffic Assignment for Public Transit Networks. In C. S. Iliopoulos, S. P. Pissis, S. J. Puglisi, and R. Raman, editors, *16th International Symposium on Experimental Algorithms (SEA 2017)*, volume 75 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20:1–20:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/LIPIcs.SEA.2017.20`.

2 O. Cats. *Dynamic Modelling of Transit Operations and Passenger decisions*. PhD thesis, KTH Royal Institute of Technology, 2011.

3 O. Cats and M. Hartl. Modelling public transport on-board congestion: comparing schedule-based and agent-based assignment approaches and their implications. *Journal of Advanced Transportation*, 50(6):1209–1224, 2016. `doi:10.1002/atr.1398`.

4 O. Cats and J. West. Learning and adaptation in dynamic transit assignment models for congested networks. *Transportation Research Record: Journal of the Transportation Research Board*, 2674:113–124, 2020. `doi:10.1177/0361198119900138`.

5 O. Cats, J. West, and J. Eliasson. A dynamic stochastic model for evaluating congestion and crowding effects in transit systems. *Transportation Research Part B Methodological*, 89:43–57, June 2016. `doi:10.1016/j.trb.2016.04.001`.

6 Z. Christoforou, E. Chandakas, and I. Kaparias. Investigating the impact of dwell time on the reliability of urban light rail operations. *Urban Rail Transit*, 6:116–131, 2020. `doi:10.1007/s40864-020-00128-1`.

7 J. Dibbelt, T. Pajor, B. Strasser, and D. Wagner. Connection scan algorithm. *ACM J. Exp. Algorithmics*, 23, October 2018. `doi:10.1145/3274661`.

8 Collection of open source public transport networks by DFG Research Unit “FOR 2083: Integrated Planning For Public Transportation”, 2018. URL: `https://github.com/FOR2083/PublicTransportNetworks`.

9 Q. Fu, R. Liu, and S. Hess. A review on transit assignment modelling approaches to congested networks: A new perspective. *Procedia - Social and Behavioral Sciences*, 54:1145–1155, 2012. `doi:10.1016/j.sbspro.2012.09.829`.

10 G. Gentile, M. Florian, Y. Hamdouch, O. Cats, and A. Nuzzolo. The theory of transit assignment: Basic modelling frameworks. In G. Gentile and K. Noekel, editors, *Modelling Public Transport Passenger Flows in the Era of Intelligent Transport Systems: COST Action TU1004 (TransITS)*, chapter 6, pages 287–386. Springer International Publishing, 2016. `doi:10.1007/978-3-319-25082-3_6`.

11 G. Gentile, K. Noekel, J.-D. Schmöcker, V. Trozzi, and E. Chandakas. The theory of transit assignment: Demand and supply phenomena. In Guido Gentile and Klaus Noekel, editors, *Modelling Public Transport Passenger Flows in the Era of Intelligent Transport Systems: COST Action TU1004 (TransITS)*, chapter 7, pages 387–481. Springer International Publishing, 2016. doi:10.1007/978-3-319-25082-3_7.

12 PTV Group. New features at a glance — PTV Visum 2024. www.ptvgroup.com, 2023.

13 Y. Hamdouch and S. Lawphongpanich. Schedule-based transit assignment model with travel strategies and capacity constraints. *Transportation Research Part B: Methodological*, 42:663–684, August 2008. doi:10.1016/j.trb.2007.11.005.

14 Y. Hamdouch, W. Szeto, and Y. Jiang. A new schedule-based transit assignment model with travel strategies and supply uncertainties. *Transportation Research Part B: Methodological*, 67:35–67, September 2014. doi:10.1016/j.trb.2014.05.002.

15 M. Müller-Hannemann and R. Rückert. Dynamic event-activity networks in public transportation: Timetable information and delay management. *Datenbank-Spektrum*, 17, May 2017. doi:10.1007/s13222-017-0252-y.

16 S. Nguyen and S. Pallottino. Equilibrium traffic assignment for large scale transit networks. *European Journal of Operational Research*, 37(2):176–186, November 1988. doi:10.1016/0377-2217(88)90327-X.

17 O. Nielsen and R. Frederiksen. Optimisation of timetable-based, stochastic transit assignment models based on MSA. *Annals OR*, 144:263–285, April 2006. doi:10.1007/s10479-006-0012-0.

18 A. Nuzzolo, U. Crisalli, and L. Rosati. A schedule-based assignment model with explicit capacity constraints for congested transit networks. *Transportation Research Part C Emerging Technologies*, 20, February 2012. doi:10.1016/j.trc.2011.02.007.

19 A. Nuzzolo, F. Russo, and U. Crisalli. A doubly dynamic schedule-based assignment model for transit networks. *Transportation Science*, 35:268–285, August 2001. doi:10.1287/trsc.35.3.268.10149.

20 N. Oliker and S. Bekhor. A frequency based transit assignment model that considers online information and strict capacity constraints. *EURO Journal on Transportation and Logistics*, 9(1):100005, 2020. doi:10.1016/j.ejtl.2020.100005.

21 M. Rieser, D. Grether, and K. Nagel. Adding mode choice to multiagent transport simulation. *Transportation Research Record: Travel Demand Forecasting 2009*, 2132:50–58, December 2009. doi:10.3141/2132-06.

22 P. Ryus, A. Danaher, M. Walker, F. Nichols, B. Carter, E. Ellis, L. Cherrington, and A. Bruzzone. *Transit Capacity and Quality of Service Manual, Third Edition.* The National Academies Press, January 2013. doi:10.17226/24766.

23 P. Schiewe, A. Schöbel, S. Jäger, S. Albert, U. Baumgart, C. Biedinger, V. Grafe, S. Roth, A. Schiewe, F. Spühler, M. Stinzendörfer, and R. Urban. LinTim: an integrated environment for mathematical public transport optimization. Documentation for version 2023.12, 2023. URL: https://nbn-resolving.de/urn:nbn:de:hbz:386-kluedo-75699.

24 J.-D. Schmöcker, A. Fonzone, H. Shimamoto, F. Kurauchi, and M. Bell. Frequency-based transit assignment considering seat capacities. *Transportation Research Part B: Methodological*, 45:392–408, February 2011. doi:10.1016/j.trb.2010.07.002.

25 H. Spiess and M. Florian. Optimal strategies: A new assignment model for transit networks. *Transportation Research Part B: Methodological*, 23:83–102, April 1989. doi:10.1016/0191-2615(89)90034-9.

26 M. Wahba. *MILATRAS: MIcrosimulation Learning-based Approach to TRansit ASsignment.* PhD thesis, University of Toronto, 2008.

27 M. Wahba and A. Shalaby. Learning-based framework for transit assignment modeling under information provision. *Transportation*, 41, March 2014. doi:10.1007/s11116-013-9510-5.

28 M. Wardman and G. Whelan. Twenty years of rail crowding valuation studies: Evidence and lessons from british experience. *Transport Reviews*, 31:379–398, May 2011. doi:10.1080/01441647.2010.519127.